ioProgrammo Anno IX - N°8 (94)

ASP.NET 2.0: TUTTE LE NOVITÀ
IL NUOVO FRAMEWORK È IN ARRIVO
ECCO COME TI AIUTERÀ NEL TUO LAVORO

Rivista + Le Grandi Guide di ioProgrammo n° 2 a € 14,90 in più

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 VERSIONE STANDARD RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • SETTEMBRE 2005 • ANNO IX, N.8 (94)

C# & MICROSOFT MAPPOINT

UNO STRADARIO NEI TUOI PROGRAMMI

- Scrivi subito il codice per visualizzare due punti su una mappa
- Impara a usare le funzioni per calcolare distanze e determinare percorsi stradali
- Pianifica i movimenti di mezzi, persone, merci e fai crescere la tua azienda

INTEGRATED PERFORMANCE PRIMITIVES

CPU DA POLE POSITION? ECCO LE LIBRERIE GIUSTE

Sfrutta tutta la potenza della macchina sviluppando software su misura per il processore

JAVOLUTION!!! IL REAL TIME È ARRIVATO

Le tue applicazioni Java fino al 70% più veloci. Riutilizzo degli oggetti e multitasking: tutti i segreti

IOPROGRAMMO WEB

IIS RIPROGRAMMATO

Intercetta le richieste del browser e "manovrale" a modo tuo prima che arrivino al server

VIDEOGAMING

REALE O VIRTUALE?

Scopri il Normal Mapping, la tecnica per ottenere simulazioni così perfette da sfiorare la realtà

TUTORIAL

HELPDESK REALIZZALO IN JAVA

Legge le richieste dall'email genera un ticket di assistenza e organizza il supporto ai clienti

CRYSTAL REPORT PRIMI PASSI

Impara a creare rapporti efficaci con lo strumento integrato in Visual Studio

DATABASE OPENSOURCE

POSTGRESQL 8.0 STENDE TUTTI!

Dalle Stored Procedure alle Foreign keys, usa le funzioni che altri DB non hanno

.NET HOW TO

SALVA LA CONFIGURAZIONE

Il configuration block: ecco come puoi conservare le impostazioni su file INI o su altri supporti

IMPOSTA LE OPZIONI

Replica la griglia delle proprietà di Visual Studio nei tuoi software

JAVA

NASCONDI IL CODICE

Le tecniche per impedire che i malintenzionati disassemblino i tuoi programmi

PDF: CREALI E PROTEGGILI

Come consentire ai tuoi programmi Java di generare report e proteggerli da chi non è autorizzato

ED ANCORA C++ ED OTL: GESTISCI FACILMENTE I DATABASE
VISUAL BASIC E I DVD: CREA UN TUO PLAYER PER IL VIDEO
JAVA E PBEANS: PERSISTENZA DEI DATI.. MA CON SEMPLICITÀ

LOCALIZZAZIONE GEOGRAFICA CON C# e MAPPOINT • INTEGRATED PERFORMANCE PRIMITIVES • REAL TIME IN JAVA CON JAVOLUTION

EDIZIONI MASTER www.edmaster.it

Anno IX - N.ro 8 (94) - Settembre 2005 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Redazione: Raffaele del Monaco, Fabio Farnesi
Collaboratori: R. Allegra, M. Autiero, D. Berta, M. Bigatti, D. Bochicchio, L. Buono, F. Cozzolino, F. D. De Michelis, F. C. Ferracchiati, F. Fortino, F. Grimaldi, M. Locuratolo, F. A. Marroccelli, L. Mattei, F. Paparoni, A. Pelleriti, M. Poponi, E. Somma, E. Tavolaro, A. Trapani, F. Vaccaro.
Segreteria di Redazione Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale: SET S.r.l.
Coordinamento Tecnico: Piero Mannelli
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Tel. 02 831213 - Fax 02 83121330
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 20% sul prezzo di copertina di €75,90
Offerte valide fino al 30/09/05
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelliera, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Neotek S.r.l.- C.da Imperatore - zona ASI Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Agosto 2005

Edizioni Master edita: Computer Bild, Idea Web, Go!OnLine Internet Magazine, Win Magazine, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, Filmteca in DVD, La mia videoteca, TV e Satellite, Win Extra, Home entertainment, Digital Japan, Digital Music, Horror Mania, ioProgrammo Extra, I mitici all'italiana, Win Junior, PC Junior, Guide strategiche di Win Magazine Giochi, Japan Cartoon, Le collection

Il fascino della sperimentazione

Presi dai ritmi frenetici della produzione, dalle scadenze imminenti, dalle richieste pressanti dei clienti, capita spesso a noi programmatori di dimenticare il motivo per cui lo siamo diventati.
A muovere l'interesse verso un mondo così complesso come quello dello sviluppo è stata probabilmente la passione della sfida. Il trovarsi di fronte a un problema e risolverlo attraverso una sequenza di passi logici è stato probabilmente il motivo per cui molti di noi programmatori hanno passato notti insonni riflettendo davanti a un monitor. A raccontarlo a chi non appartiene alla "comunità dei programmatori" fa probabilmente sorridere, o suscita un sorriso divertito, eppure è questo nostro modo di essere "hacker" che ci ha condotto a fare della nostra passione anche il nostro mestiere. Certo siamo hacker, lo siamo tutti, non inteso nel senso di pirati ma intesi come persone che cercano ogni volta il proprio limite di conoscenza, lo abbattono e si pongono un altro limite da raggiungere e superare. E' il fascino perverso della conoscenza che ci conduce ad essere bravi programmatori. Per una volta dunque voglio non concentrare la mia attenzione su un editoriale "tecnico" che illustri il punto della situazione nel campo dello sviluppo, voglio invece esortarvi a sperimentare, a farvi trasportare da quella voglia di conoscere che è anche il motivo della nostra passione. Solo così riusciremo a creare software innovativi e a muovere un mondo dello sviluppo che ha bisogno prima di ogni cosa del nostro entusiasmo.

Fabio Farnesi

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre \soft\codice\ e \soft\tools\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

METTI UN GPS NEI TUOI PROGRAMMI

CREA UN SOFTWARE CHE, DATI DUE INDIRIZZI, NE MOSTRA LA POSIZIONE SU UNA MAPPA, NE CALCOLA LA DISTANZA, DETERMINA IL PERCORSO STRADALE CHE LI COLLEGA E MOLTO ALTRO ANCORA...

- Da utilizzare per programmare un viaggio
- Per gestire piccole flotte di rappresentanti in movimento
- Da integrare in applicazioni gestionali e di logistica

Via dei Lucchesi

pag. 14

PDF? SI GRAZIE!!! FACCIAMOLO CON JAVA

Come consentire ai propri programmi Java di generare report in formato PDF e proteggerli da chi non è autorizzato pag. 32

http://forum.ioprogrammo.it

RUBRICHE

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

RIVISTA + LIBRO + CD-ROM in edicola

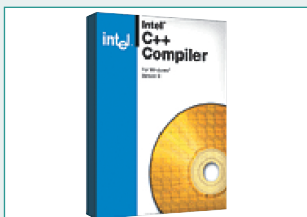
RIVISTA + CD-ROM in edicola

Prodotti del mese

Intel C++ Compiler 9.0

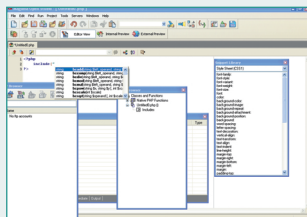
Il compilatore che esaspera la potenza del tuo processore!

Molte aree della programmazione, come il calcolo scientifico, la manipolazione real-time dei segnali e delle immagini e l'Intelligenza Artificiale, hanno in comune la necessità di prestazioni molto elevate. Diventa indispensabile compilare per uno specifico modello di processore, traendo vantaggio dalla possibilità di appoggiarsi su specifiche sicure e avanzate. Il compilatore Intel é ottimizzato per i processori dell'omonima piattaforma, e produce un codice compilato che sfrutta la potenza dei processori fino all'ultimo bit. Per utilizzare il prodotto è necessario registrarsi, la licenwa é valida per 30 giorni.

[pag.123]

Maguma Open Studio 1.0

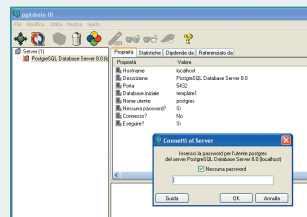
L'editor PHP che ti semplifica la vita. Anche con Debugger

Si tratta, probabilmente il miglior IDE OpenSource disponibile, ovvero un ambiente di sviluppo che aiuta il programmatore nella creazione del proprio codice. Le caratteristiche di Maguma Open Studio che semplificano la vita del programmatore sono molteplici. È dotato di Code Completion e Sintax Highlighting che sono sempre la base per ogni IDE che si rispetti. Oltre a queste due caratteristiche comuni a molti ambienti di programmazione ce ne sono molte altre che fanno di Magma un ottimo IDE. Ad esempio la possibilità di gestire direttamente le connessioni PHP e modificare direttamente i file in remoto.

[pag.123]

PostgreSQL 8.0

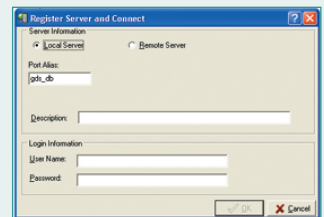
Leggero, professionale, gratuito, un caposaldo nel suo genere

Per certi versi PostgreSQL ha subito nel corso del tempo gli attacchi sferrati dai concorrenti MySQL e Firebird per primi, tuttavia ha conservato intatta tutta la sua base di installato. I motivi di questa stabilità sono insiti nelle caratteristiche del database: solido come una roccia, veloce oltre ogni aspettativa Postgres è usatissimo sia in applicazioni web che in applicazioni standalone. Probabilmente fino a qualche tempo fa soffriva del fatto di essere multipiattaforma ma con un occhio speciale rivolto a Linux, il che lo rendeva particolarmente complicato da installare in ambienti Windows. Questa limitazione é stata ampiamente superata.

[pag.119]

Borland Interbase 7.5

Il DataBase che ha fatto la storia. Completo e affidabile!

Nato con le prime versioni di Delphi, compatibile SQL, multiDb, multithreading quando ancora tutti usavano db personalizzati e i più evoluti al massimo Access, Interbase ha fatto la storia della programmazione. Rilasciato sotto licenza OpenSource nel periodo in cui Borland è diventata Imprise salvo poi ritornare un software commerciale in tempi successivi è senza dubbio un database dalle prestazioni straordinarie, che probabilmente non è diventato un leader solo a causa dell'altalenanza di situazioni in cui la sua politica di Marketing si è venuta sviluppare.

[pag.118]

RILASCIATE LE QT 4.0

Trolltech la software house, sviluppatrice delle note librerie per lo sviluppo di applicazioni multipiattaforma ne ha appena rilasciato la versione 4.0. Fra le novità più interessanti una maggiore integrazione con Visual Studio .NET che dovrebbe rendere semplificato lo sviluppo di applicazioni dedicate ad ambienti Business. Erik Chambe-Eng presidente di Trolltech ha dichiarato che lo sviluppo di applicazioni rivolte ad una molteplicità di sistemi operativi ha assunto ormai un'importanza molto al di là delle aspettative. La versione 4.0 delle QT consente di scrivere applicazioni sufficientemente performanti in qualsiasi ambiente e in qualsiasi settore, dal multimedia alla grafica a programmi basati sull'uso massiccio di database che necessitano anche di una solida infrastruttura di tipo server.

ARRIVA GOOGLE VIDEO

E con questo la gamma delle informazioni ricercabili con Google è completa. Mai dire mai con Google, ma realmente il nuovo servizio in una qualche maniera chiude il cerchio dei dati reperibili attraverso il noto motore di ricerca. Da qualche tempo è possibile accedere all'indirizzo *http://video.google.com/*, digitare una parola, oppure utilizzare ricerche più complesse per ottenere un elenco di video ad alta definizione che riguardano l'argomento.
L'unica nota stonata rispetto a questo servizio è che il player è attualmente disponibile nella sola versione Windows, ma essendo il tutto ancora in versione Beta ci sono i margini per recuperare.

GRANDE SUCCESSO PER THE SUMMER OF CODE

Google si sta rivelando una delle compagnie più attive al mondo nel campo dell'innovazione tecnologica. Se da un lato gli annunci di nuovi servizi si susseguono in maniera continuativa e frenetica dall'altro il motore di ricerca più famoso al mondo non cessa di investire in nuovi progetti. Il caso di Summer of Code è emblematico di quanto innovativa sia Google. Summer of Code è un progetto nato per fare avvicinare gli studenti al mondo dell'OpenSource.
L'innovazione sta nel fatto che ciascuno studente riconosciuto idoneo alla partecipazione al progetto verrà retribuito con uno stipendio di 5000$. Dal totale di questi 5000 ne verranno detratti 500, da destinarsi alle compagnie “mentors” del progetto. Una compagnia mentors è una società che ospiterà gli studenti per lavorare ai propri progetti. Il Summer of Code realizza così un duplice obiettivo, da un lato mette a disposizione delle compagnie che sviluppano OpenSource dei bravi programmatori, dall'altro mette i bravi programmatori in grado di fare esperienza su un progetto concreto e per giunta retribuisce i loro sforzi con uno

MANDRIVA SCALATA AL POTERE

Gli analisti dicono che Linux non sia pronto per il Desktop, nonostante la quantità di installazioni del sistema del pinguino anche sui PC degli utenti comuni sia in aumento. Si è vero, forse dovrà passare ancora del tempo prima che le piccole e medie aziende diano in mano ai loro dipendenti un sistema Linux, tuttavia c'è chi ci crede e lo fa con tutte le proprie forze. A guidare la carovana delle società che stanno investendo in Linux come ambiente pronto per il desktop è Mandriva, ex Mandrake. La società di Gael Duval dopo avere acquisito Conectiva appena qualche mese fa, ha rilevato adesso anche le quote di Lycoris. In tutti e due i casi si tratta di acquisizioni di società che hanno incentrato la propria strategia nel creare strumenti tali da rendere Linux un sistema semplice da usare e alla portata anche dell'utente meno smaliziato. Allo stesso modo Mandrake fin dalla sua nascita insegue il sogno di proporsi sui sistemi Desktop come reale alternativa a Microsoft Windows. Per molti la visione di un pinguino sui desktop di casalinghe e segretarie d'azienda potrebbe essere ancora un miraggio, ma Mandrake dimostra invece di credere fermamente in questa possibilità, d'altra parte gli innovatori sono sempre un po' tacciati di essere dei visionari, ma poi alla fine non sono i visionari che creano le rivoluzioni più importanti?

PRIMI FORK PER OPENSOLARIS

Sun non ha quasi fatto in tempo a rendere disponibile il codice sorgente di OpenSolaris che c'è già pronta un fork! Si chiama ShilliX ed è la prima distribuzione nata dal codice del sistema operativo di Sun. Da un punto di vista strettamente tecnologico la novità sta nel fatto che l'intero sistema può risiedere si una chiavetta USB, ma in realtà la novità più grande non è solo quella tecnica ma soprattutto quella filosofica. Il cambio di direzione imposto da Sun prima con Java e poi con OpenSolaris conferma in pieno l'idea di innovazione che sta alla base dell'OpenSource. Il codice sorgente di un software dovrebbe essere sempre disponibile perché la sua disponibilità innalza la possibilità che qualcuno apporti delle innovazioni e questo non può che fare bene al progresso.

VENTUNO GB IN UN SOLO CM^2

È questo il risultato raggiunto da alcuni ricercatori dell'università di Tohoku in Giappone. È un'innovazione significativa che da un lato abbassa il costo delle periferiche di storage dei dati, dall'altro innalza il livello tecnologico dei sistemi embedded che hanno raggiunto dimensioni talmente piccole da rappresentare un vero ufficio portatile. Un hd da un cm^2 in grado di contenere 21GB di dati sarebbe ben accetto su qualunque palmare.

stipendio di tutto rispetto. Il successo del Summer Of Code al momento in cui scriviamo è stato enorme, tanto che il numero di studenti accettati dal progetto è stato elevato da 200 a 400 con un corrispondente innalzamento dei fondi stanziati per lo scopo. Il numero di studenti che hanno fatto richiesta si aggira intorno ai 9000 in pochissimi giorni, tanto che Google ha dovuto smettere di accettare richieste e si è riservato di rispondere a tutti in 10 giorni lavorativi. Che dire? Un peccato per chi non è riuscito ad approfittare di questa opportunità, un enorme in bocca al lupo ai 400 fortunati che faranno questa grande esperienza, aspettiamo di conoscerne gli esiti...

LA NORVEGIA DICE NO!

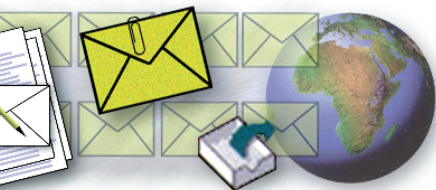
Epoca particolarmente strana la nostra, in cui l'OpenSource e la forza dirompente del Free Software aprono scenari inimmaginabili solo qualche anno fa. Non stupisce dunque questa dichiarazione di Morten Andreas Meyer, ministro per la modernizzazione norvegese secondo il quale il proprio governo non dovrebbe adottare formati proprietari nelle proprie comunicazione. La dichiarazione è di quelle pesanti, immaginate infatti cosa succederebbe in Italia se venisse proibito negli uffici pubblici l'uso di Word o di Excel! D'altra parte la Norvegia non è nuova a questo genere di iniziative è da sempre si è mostrata piuttosto all'avanguardia nel campo informatico. Il piano presentato da Meyer prende il nome di eNorge 2009 e prevede un graduale passaggio di tutta la pubblica amministrazione a software OpenSource. Ora, la questione si fa interessante, da un lato non crediamo che Microsoft abbia voglia di perdere un cliente importante come l'amministrazione norvegese, dall'altro è un bel rischio che un governo statale potrebbe correre.
A dire il vero fin qui le dichiarazioni di vari governi si sono succedute in questo senso senza poi dare un seguito concreto a quanto proclamato.
D'altra parte viviamo in un'epoca di grandi incertezze su temi quali il copyright e il giusto modello di business per la distribuzione del software si sviluppano in direzioni del tutto inaspettate.

IN ARRIVO LA SESTA VERSIONE PER DIVX

DivX, il popolare formato di compressione spauracchio di case cinematografiche e musicali ha raggiunto la versione numero sei. C'è da dire che se in precedenza già in molti hanno scatenato guerre senza precedenti all'uso del DivX, questa volta la nuova release contiene novità tali da farlo diventare un rivale estremamente pericoloso per chiunque voglia proporre standard diversi per la compressione video.
Alla tradizionale qualità affiancata però da un alto livelli di compressione si è aggiunta infatti l'interattività. DivX 6 supporta l'uso dei menu, la divisione in scene, i sottotitoli e persino il multilingue, tutte caratteristiche che lo rendono idoneo ad essere utilizzato in più di un'occasione. Con questa struttura il DivX si presenterebbe infatti come un formato ideale per essere utilizzato in periferiche di riproduzione video.

La lotta sarà ardua e certamente dalla fine del 2005 in poi ci saranno scintille fra i produttori tradizionali e chi tenterà di sfruttare il DivX in qualunque modo.
In ogni caso certamente non è una colpa da attribuire alla tecnologia, quanto a un modo di intendere la cessione dei diritti che nella nostra epoca suscita non poche discussioni e coinvolge più di un settore, non solo quello relativo a musica e a film.

QUASI PRONTO IL FILE SHARING DI MICROSOFT

Secondo Christos Gkantsidis e Pablo Rodriguez due ricercatori dell'università di Cambridge, BitTorrent e soci non sarebbero sufficientemente ottimizzati. Di fatto i normali client P2P soffrono del problema dello Swarming. Si tratta di un rallentamento delle prestazioni che si verifica quando il numero degli utenti contemporanei che stanno ricevendo un file diventa elevato. Il problema nascerebbe da una certa complessità dalla gestione degli algoritmo di scheduling. Cosi' a Christos Gkantsidis e Pablo Rodriguez si sono aggiunti Phil Chou e Kamal Jain di Microsoft che hanno dato vita al progetto Avalanche, ovvero un sistema che propone un diverso approccio alla filosofia del File Sharing migliorandolo negli aspetti più deboli. Microsoft dunque potrebbe proporre il proprio sistema di File Sharing, ma chi si sentirà di condividere i propri file sotto l'egida del gigante di Redmond?

NEWSGROUP Le informazioni nella Rete

Direttamente dal forum di ioProgrammo http://forum.ioprogrammo, le discussioni più "Hot" del momento

MySQL

MySQL e chiavi esterne

**Volevo qualche chiarimento sulle chiavi esterne di MySQL.
Fino ad ora ho creato le tabelle "collegandole" logicamente tramite le chiavi primarie e specificando tali "collegamenti" nelle query (... WHERE t1.id_a=t2.id_b).
Mi domando come specificare le chiavi esterne (ho visto qualche esempio, ma non mi è proprio chiara la sintassi) e quali vantaggi mi darebbero nel maneggiare le tabelle collegate! Grazie a tutti anticipatamente**

MauroXYZ7

http://forum.ioprogrammo.it/thread.php?threadid=6152&boardid=15

Risponde giamig

Considera il seguente codice:

```
CREATE TABLE `cliente` (
 `COGNOME_NOME` varchar(100) NOT NULL
   default '', `RAGIONE_SOCIALE` varchar(100) default '', `CODICE_FISCALE`
   varchar(16) NOT NULL default '', `CODICE`
   varchar(100) NOT NULL default '', PRIMARY
   KEY (`CODICE`) ) TYPE=InnoDB;
CREATE TABLE `fattura` (
   `CODICE_FATTURA` int(11) NOT NULL
   auto_increment, `CODICE_CLIENTE` varchar(100) NOT NULL default '', PRIMARY KEY
   (`CODICE_FATTURA`), KEY `COD_CLI`
   (`CODICE_CLIENTE`), CONSTRAINT `0_27`
   FOREIGN KEY (`CODICE_CLIENTE`) REFERENCES `cliente` (`CODICE`) ON DELETE
   CASCADE ON UPDATE CASCADE
   ) TYPE=InnoDB;
```

Nella tabella fattura c'è una chiave esterna (FOREIGN KEY), cioè il campo CODICE_CLIENTE che si riferisce a cliente (CODICE). Il tuo modo di "collegare" le tabelle funziona.. ma non è il miglior metodo.
Un minimo errore nella query e rischi di fare la frittata e rendere il db inconsistente.
Il vincolo di chiave esterna inserito nella fattura non permette di memorizzare una fattura di cui non esiste il cliente corrispondente nel db.
Inoltre nel caso in cui il cliente venga modificato o addirittura cancellato dal db anche la fattura dovrà essere modificata o cancellata: per questo si scrive "ON DELETE CASCADE" e "ON UPDATE CASCADE".

Java

Come avere sempre l'ultima riga di un JScroll

Devo aggiungere dei messaggi a un JTextArea che è contenuto in un Jscroll. Come faccio a visualizzare sempre l'ultima riga? Intendo quella con il messaggio più recente.

Derr

http://forum.ioprogrammo.it/thread.php?threadid=6127&boardid=18

Risponde Giamig

Prova con qualcosa del tipo *text.setCaretPosition(text.getText().length)*...

Copiare i file da una cartella a un'altra

**Potete postarmi gentilmente il codice per far copiare tutti i file da una cartella a un'altra (già esistente).
Grazie**

matdev

http://forum.ioprogrammo.it/thread.php?threadid=6150&boardid=18

Risponde Giamig

Questo metodo copia tutto il contenuto di una cartella (pathSorgente) ed eventuali sottocartelle nella cartella di destinazione (pathDestinazione).
La copia delle sottocartelle avviene ricorsivamente.

```
private void copiaFile(String pathSorgente,
                    String pathDestinazione){
 File dirSorgente=new File(pathSorgente);
 File[] filePresenti=dirSorgente.listFiles();
 File dirDestinazione=new
                      File(pathDestinazione);
 File fileDestinazione;
 try {
   dirDestinazione.mkdir();
   for (int i = 0; i < filePresenti.length;
                                    i++) {
     if (filePresenti[i].isFile()) {
       //per ogni file della lista effettuo
         una copia nella dir di destinazione
     in = new FileInputStream(
                        filePresenti[i]);
     br = new BufferedInputStream(in);
     fileDestinazione=new File(
       dirDestinazione.getAbsolutePath()+
           "\\"+filePresenti[i].getName());
     fileDestinazione.createNewFile();
     out = new FileOutputStream(
                        fileDestinazione);
     bw = new BufferedOutputStream(
                                   out);
     while (((lineaLetta = br.read()) !=
                                   -1)) {
     //copio nel file di destinazione
     bw.write(lineaLetta); }
   br.close();
   bw.close();
   in.close();
   out.close(); }
 else{
   //Se si tratta di una directory chiamo
     ricorsivamnete la procedura di "copia"
                           sui nuovi path
   copiaFile(dirSorgente.getAbsolutePath()+
```

```
"\\"+filePresenti[i].getName()
,dirDestinazione.getAbsolutePath()+
"\\"+filePresenti[i].getName()); }
  }
}
catch (IOException ex) {
  System.err.println("ERRORE:
  +"ex.getMessage()); }
}
```

C#

Trasferimento immagine in rete

Vorrei creare un applicazione server ed una client... il server deve inviare al client un immagine (memorizzata in un oggetto bitmap) e il client deve riceverla e mostrarla in una pictureBox... Come fare?

yperalex

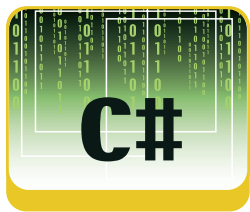
http://forum.ioprogrammo.it/thread.php?threadid=6100&boardid=29

Risponde Salvatore Meschini

Nell'ipotesi che il server sia un webserver fai riferimento a questo tip.

```
// Il progetto è compilabile da riga di
// comando con csc.exe
// csc /target:winexe nomefile.cs
// Il codice mostra come visualizzare
// un'immagine in un controllo PictureBox
// scaricandola da Internet (in modalità
// multithread). In questo modo l'utente
// può interagire con l'applicazione anche
// durante il download delle immagini...
using System;
using System.Windows.Forms;
using System.Net;
using System.Drawing;
using System.Threading;
namespace MyFormProject
{ class MainForm :
  System.Windows.Forms.Form
{ private System.Windows.Forms.TextBox
  Indirizzo;
private System.Windows.Forms.Button
  CaricaBtn;
private System.Windows.Forms.PictureBox
  MioPicBox;
public MainForm()
{ InitializeComponent(); }
void InitializeComponent() {
this.MioPicBox = new System.Windows
  .Forms.PictureBox();
this.CaricaBtn = new System
  .Windows.Forms.Button();
this.Indirizzo = new System.Windows.
  Forms.TextBox();
this.SuspendLayout();
//
// MioPicBox
//
this.MioPicBox.Location = new
  System.Drawing.Point(24, 16);
this.MioPicBox.Name = "MioPicBox";
this.MioPicBox.Size = new
  System.Drawing.Size(248, 136);
this.MioPicBox.TabIndex = 0;
this.MioPicBox.TabStop = false;
//
// CaricaBtn
//
this.CaricaBtn.Location = new
  System.Drawing.Point(104, 192);
this.CaricaBtn.Name = "CaricaBtn";
this.CaricaBtn.TabIndex = 1;
this.CaricaBtn.Text = "Carica";
// Gestore evento "Clicca sul tasto"
this.CaricaBtn.Click += new
  System.EventHandler(this.CaricaBtnClick);
//
// Indirizzo
//
this.Indirizzo.Location = new
  System.Drawing.Point(16, 160);
this.Indirizzo.Name = "Indirizzo";
this.Indirizzo.Size = new
  System.Drawing.Size(256, 20);
this.Indirizzo.TabIndex = 2;
this.Indirizzo.Text = "http://localhost/apa-
che_pb.gif";
//
// MainForm
//
this.AutoScaleBaseSize = new
  System.Drawing.Size(5, 13);
this.ClientSize = new
  System.Drawing.Size(292, 230);
this.Controls.Add(this.Indirizzo);
this.Controls.Add(this.CaricaBtn);
this.Controls.Add(this.MioPicBox);
this.Name = "MainForm";
this.Text = "Carica Immagine da URL";
this.ResumeLayout(false); }
[STAThread]
public static void Main(string[] args)
{
  Application.Run(new MainForm());
}
// Classe per la gestione del download
  (accetta parametri)
public class ThreadScaricamento
{ private string pURL;
  private PictureBox pPicBox;
  private bool Esito = false;
  // Imposta i parametri (PictureBox per
  riferimento)
public ThreadScaricamento(ref PictureBox
  PicBox, string URL)
{
  pURL = URL;
  pPicBox = PicBox;
}
// Esegue le operazioni più importanti
public void Scarica()
{  try
{  // Prova a caricare l'immagine da un URL
pPicBox.Image = Image.FromStream(
 ((WebRequest.Create(pURL)).GetResponse()
  ).GetResponseStream());
}
catch (WebException)
// Si possono gestire anche altre eccezioni
{ pPicBox.Image = null; // In caso di errore
}
Esito = (pPicBox.Image != null); // True o
  False
HoFinito();
}
// Il metodo viene richiamato quando il
  thread termina
// Da modificare in base alle proprie esigenze
private void HoFinito()
{  if (Esito)
MessageBox.Show("TUTTO OK!"); else
MessageBox.Show("ERRORE!");
}
} // Fine Classe ThreadScaricamento
// Alla pressione del tasto:
void CaricaBtnClick(object sender,
  System.EventArgs e)
{ if (Indirizzo.Text != "") // Indirizzo è un
  controllo EditBox
{  // Passa i parametri
ThreadScaricamento TS = new
  ThreadScaricamento(ref MioPicBox,
  Indirizzo.Text);
Thread T = new Thread(new
 ThreadStart(TS.Scarica)); // Crea il Thread
T.Start(); // Avvia il thread
  }
else MessageBox.Show("Devi fornire un
  indirizzo...");
  }
 }
}
```

Per non perdere mai la bussola!

Chi di voi non ha mai realizzato un'anagrafica clienti alzi la mano! In questo articolo ne realizzeremo una con una marcia in più. Visualizzeremo la posizione degli uffici del cliente su una mappa!

Fino a qualche anno fa, quando avevamo la necessità di raggiungere un luogo, ci affidavamo alle onnipresenti "cartine" stradali. Chi di voi non si è fermato almeno una volta in autostrada ad acquistarne una? Muoversi in città era poi ancora più complesso: cartina molto più "densa", nomi delle strade scritti con caratteri piccolissimi ed il famoso indice che indicava, per ogni strada, pagina e quadrante della mappa. Decisamente non era il massimo della comodità! Fortunatamente per noi quei tempi sono finiti. Per andare da un qualsiasi luogo del pianeta ad un altro o semplicemente per trovarne uno, apriamo il nostro browser preferito, andiamo su uno dei tanti siti specializzati, cerchiamo l'indirizzo ed il problema è risolto!

MAPPOINT

MapPoint è il sistema sviluppato da Microsoft per la localizzazione geografica. Questo prodotto è distribuito sostanzialmente in tre versioni diverse adatte a scopi specifici. Vediamole brevemente:

- **MapPoint Web Service:** è un servizio web che espone sostanzialmente il motore di MapPoint. Attraverso l'utilizzo di questo servizio è possibile realizzare applicazioni personalizzate per la gestione della localizzazione geografica senza preoccuparsi di installare altri prodotti lui PC dei nostri clienti.

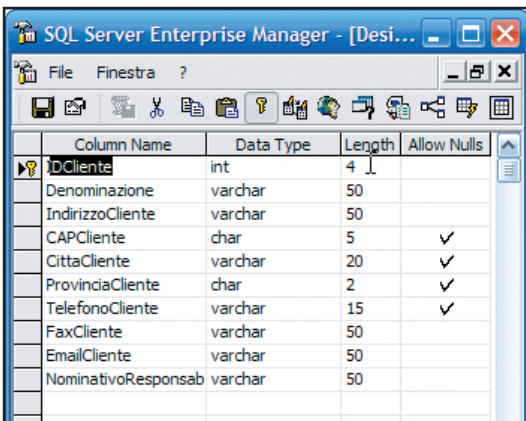
- **MapPoint Location Server:** è un sistema di localizzazione real time che sfrutta dispositivi mobili e MapPoint Web Service per la localizzazione immediata di entità in movimento.

- **MapPoint 2004:** è la versione software di MapPoint. Questa versione è stata realizzata sia per poter essere utilizzata direttamente, sia per poter visualizzare informazioni provenienti da file del pacchetto office.

Escludendo la versione *MapPoint Location Server* (la cui implementazione esula dallo scopo di questo articolo), per un'applicazione standard abbiamo la possibilità di scegliere tra la versione *MapPoint Web Service* e la versione *MapPoint 2004*. Tale scelta deve essere fatta in modo ponderato onde evitare di sprecare i nostri soldi (o quelli dei nostri clienti).
I costi delle due versioni sono abbastanza diversi tra loro e sebbene quello della versione *client (MapPoint 2004)* è definito ($ 299), quello della versione *WebService* è abbastanza complesso da calcolare e bisogna contattare Microsoft per avere una stima precisa dei costi.
Il mio personale consiglio è:

- **MapPoint 2004:** per singole applicazioni, non multiutenza e che devono lavorare in modalità disconnessa.

- **MapPoint Web Service:** per applicazioni client-server, applicazioni web, installazioni multiple, applicazioni connesse.

Nell'applicazione descritta in questo articolo utilizzeremo la versione client di MapPoint *(MapPoint 2004)* scaricabile in demo dal sito Microsoft (vedi box laterale). Per prima cosa, va installata sulla macchina su cui abbiamo intenzione di sviluppare i nostri software e, se decidessimo di rivendere la nostra applicazione, dovrà essere installata anche sul PC dell'utente che acquista il nostro software.
La procedura di installazione è molto semplice e porta via solo pochi minuti. Una volta completato la fase di setup, possiamo iniziare lo sviluppo del nostro applicativo.

UNA SEMPLICE ANAGRAFICA

Come già detto in precedenza, quello che realizzeremo in queste pagine è una semplice anagrafica clienti a cui poi collegheremo MapPoint. Nella maggior parte dei software c'è l'esigenza concreta di memorizzare una serie di dati relativi agli utenti, clienti o fornitori. L'utilità di tali dati è tanto scontata quanto importante. L'anagrafica realizzata per questo articolo è molto semplice: archivieremo solo i dati di base dei nostri potenziali clienti e li useremo per generare la relativa mappa.
Come vedremo in seguito, lo stesso meccanismo utilizzato per i clienti sarà valido per tutte le altre entità "localizzabili" (dotate, in pratica, di un indirizzo). Il nostro Data Base di prova è composto da una sola tabella il cui schema è mostrato in **Figura 1**.

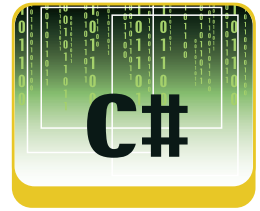
Column Name	Data Type	Length	Allow Nulls
IDCliente	int	4	
Denominazione	varchar	50	
IndirizzoCliente	varchar	50	
CAPCliente	char	5	✓
CittaCliente	varchar	20	✓
ProvinciaCliente	char	2	✓
TelefonoCliente	varchar	15	✓
FaxCliente	varchar	50	
EmailCliente	varchar	50	
NominativoResponsab	varchar	50	

Fig. 1: ***Schema della tabella clienti del DataBase***

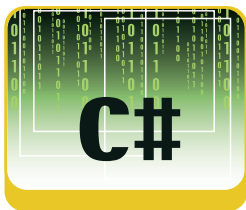
UNO SGUARDO A MAPPOINT

Facciamo il punto della situazione: abbiamo installato MapPoint, abbiamo creato il DataBase che ci servirà per inserire i dati dei nostri clienti. Ora iniziamo a sviluppare l'applicazione. Come prima cosa creiamo una nuova soluzione con Microsoft Visual Studio .net 2003 che chiameremo *ioProgrammoMap* ed aggiungiamo un riferimento alla libreria di MapPoint. Gi assembly referenziati saranno visibili nella cartella *"References"* del nostro progetto. Utilizzando la direttiva *"Using"* nel nostro codice, possiamo accedere direttamente alle funzionalità esposte. Lavorando con la versione client di questo software, dobbiamo ragionare come se le operazioni dovessimo compierle a mano: in poche parole, dobbiamo aprire il programma (MapPoint appunto), effettuare la nostra ricerca, recuperare la mappa e chiudere il programma. Questa ultima operazione è piuttosto importante in quanto, se il software non viene chiuso, ogni sua istanza resterà in memoria occupando inutilmente risorse di sistema preziose che rallenteranno inesorabilmente tutto il sistema. Cominciamo aggiungendo al nostro progetto una nuova classe che chiameremo *MapGenerator.cs*. Sarà essa ad occuparsi di gestire il nostro MapPoint; sarà in pratica il ponte tra il nostro programma e l'API di Microsoft. Analizziamo subito un metodo d'esempio che implementeremo in un secondo momento. Tale metodo dovrà generare una bitmap con la nostra mappa:

```
using System;
using System.Text;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using MapPoint;
namespace ioProgrammoMap
{
  /// Classe di gestione di MapPoint
  public class MapGenerator
  {
     private MapPoint.Map map;
     private Location location = null;
     private MapPoint.ApplicationClass app = null;
     /// Costruttore di default
     public MapGenerator(){}
      public Bitmap GetMap(){
        try{
             app = new ApplicationClass();
             app.Visible = false;
             map = new MapPoint.Map();
             map = app.ActiveMap;
             ...
             map.CopyMap();
             return BitmapFromClipboard();
        }catch (System.Runtime.InteropServices
                                  .COMException comEx){
  MessageBox.Show("Si è verificato un errore durante
        la chiamata a MapPoint!!", "ioProgrammoMap",
     MessageBoxButtons.OK, MessageBoxIcon.Error);
  }catch (Exception ex){
     MessageBox.Show("Si è verificato un errore
              nell\'applicazione!!", "ioProgrammoMap",
     MessageBoxButtons.OK, MessageBoxIcon.Error);
   }finally{
     map.Saved = true;
     if(app != null)
        app.Quit();
        app = null;
   }
   return null;
}
```

Come prima cosa, usiamo la direttiva *"Using"* per utilizzare correttamente MapPoint. Il nostro oggetto ha un costruttore di default che lasceremo momentaneamente vuoto e, subito dopo, un

On line si trovano ormai diversi siti che generano mappe per qualsiasi esigenza. Si va dall'italiano http://mappe.virgilio.it a http://maps.msn.com per arrivare al più recente http://maps.google.com completo anche di foto satellitari di tutto il territorio.

Per farsi una idea dei navigatori satellitari in commercio è sufficiente fare una ricerca usando www.google.it con chiave di ricerca "navigatori satellitari". Le soluzioni disponibili in commercio sono numerose. Si va da soluzioni dedicate e soluzioni composte da palmari e ricevitori GPS.

Utilizza questo spazio per le tue annotazioni

metodo chiamato *GetMap* che ha come tipo di ritorno una bitmap (un'immagine) con la mappa che faremo generare. Tra le prime istruzioni troviamo quella che istanzia l'oggetto *MapPoint*, la creazione di un oggetto *Map* l'attributo *app.Visible* settato a *false*. Con queste istruzioni abbiamo aperto praticamente MapPoint e lo abbiamo nascosto all'utente. Queste istruzioni, incluse quelle relative alla creazione della mappa, verranno incluse in un blocco *try/catch/finally* per la gestione strutturata delle eccezioni. Qualora una delle operazioni dovesse fallire, mostreremo all'utente un messaggio di errore specifico e, volendo, potremmo anche loggare gli errori per poterli poi analizzare successivamente. Nella parte *finally* del blocco *try/catch*, inseriamo le istruzioni idonee alla chiusura di MapPoint. In questo modo eviteremo i problemi di istanze multiple dell'applicativo di cui parlavamo in precedenza. Altro elemento molto importante di MapPoint è l'oggetto *"Location"*. Esso rappresenta un luogo ben preciso su una mappa. Se lo visualizzassimo nel *"visualizzatore oggetti"* vedremmo i suoi metodi e, soprattutto, i campi che lo caratterizzano. Individuare un punto su una mappa, per MapPoint, vuol dire essenzialmente trovare la location corrispondente e popolarne i relativi campi. Attraverso tale oggetto sarà poi possibile compiere una serie di operazioni come evidenziare il punto sulla mappa, cercare altri punti vicini ecc.

NOTA

Tutti i dettagli relativi a MapPoint possono essere recuperati dal sito ufficiale al seguente indirizzo: http://www.microsoft.com/mappoint/default.mspx. Seguendo i link relativi al servizio web sarà poi possibile attivare un account di tipo Developer per eseguire i propri test.

L'ANAGRAFICA

La sezione *Anagrafica* del nostro programma contiene i dati di base dei nostri clienti. Il campo che ci interessa maggiormente è l'indirizzo del cliente che passeremo a MapPoint. Partendo dalla considerazione che ogni anagrafica clienti, per considerarsi tale, deve contenere almeno questo valore, adattare i vostri software già pronti sarà un gioco da ragazzi. Aggiungiamo quindi al nostro progetto un nuovo form chiamato *Clienti.cs* ed una classe *Data.cs* che incapsulerà tutta la logica di accesso ai dati. All'interno di *Clienti.cs* trasciniamo un controllo *DataGrid* dalla toolbar di Visua Studio e visualizziamo il sorgente del form. Il primo metodo da implementare è relativo all'evento *load* del form e ci servirà per popolare la griglia con i dati relativi ai clienti presi dal Data Base.

```
private void Clienti_Load(object sender,
                                        System.EventArgs e) {
   dataLayer = new Data();
   dsClienti = dataLayer.dsClienti();
   dataGridClienti.DataSource = dsClienti.Tables[0];
}
```

La prima istruzione crea un'istanza dell'oggetto *Data.cs* che, come abbiamo visto, si occupa della comunicazione con il Data Base. Questo oggetto ha un metodo chiamato *dsClienti* che ritorna un dataset contenente i dati dei clienti.
Diamo quindi uno sguardo a tale oggetto:

```
public class Data
{
   private string connString;
   /// Recupero della stringa di connessione al DB
   private string ConnString{
      get{
         if ( connString == null ){
              connString = ConfigurationSettings
                                   .AppSettings["connString"];
         }
         return connString; }
   }
   /// Costruttore di default
   public Data(){}
   public DataSet dsClienti(){
      SqlConnection conn = new SqlConnection(
                                             ConnString);
      DataSet dsClienti = new DataSet("dsClienti");
      conn.Open();
      SqlDataAdapter da = new SqlDataAdapter(
                        "Select * from tblClienti", conn);
      da.Fill(dsClienti);
      return dsClienti;
   }
}
```

La prima operazione che compiamo è quella di recuperare la stringa di connessione al Data Base archiviata nel file *App.config*. Se diamo uno sguardo all'implementazione del metodo *dsClienti* ci accorgiamo che esso accede fisicamente al DataBase, legge la tabella clienti con l'istruzione *Select*, memorizza i dati estratti all'interno di un DataSet che riman-

CALCOLO DELLA DISTANZA

Il calcolo della distanza fra due punti è abbastanza semplice. L'esempio riportato da Microsoft anche su MSDN è il seguente:

```
[...]
   Set objLocNY = objMap.FindResults(
                     "New York, NY").Item(1)
   Set objLocLA = objMap.FindResults(
                   "Los Angeles, CA").Item(1)

   'Find the point on the screen
                  midway between the two
   X1% = objMap.LocationToX(
                                 objLocNY)
   Y1% = objMap.LocationToY(
                                 objLocNY)
   X2% = objMap.LocationToX(
                                 objLocLA)
   Y2% = objMap.LocationToY(
                                 objLocLA)
   x% = (X1% + X2%) / 2
   y% = (Y1% + Y2%) / 2
   Set objCenter = objMap.XYToLocation(
                                   x%, y%)
   'Show the distance
   objMap.Shapes.AddLine objLocNY,
                                 objLocLA
   Set objTextbox = objMap.Shapes
         .AddTextbox(objCenter, 200, 50)
   Distance# = objMap.Distance(
                      objLocNY, objLocLA)
   Let objTextbox.Text = "Distance,
          New York to Los Angeles: " &
                                Distance#
[...]
```

da al chiamante. Nel caso specifico il DataSet viene rimandato all'implementazione dell'evento load del form clienti. Il DataSet ricevuto viene poi associato ad un DataGrid per la visualizzazione dei dati. Il passo successivo che ci interessa implementare è quello di accedere ad un dettaglio dei dati anagrafici del cliente selezionato nel Data Grid. A tale scopo, ci viene in contro un oggetto specifico: il BindingManagerBase

```
private object GetCurrentBindedObject(DataGrid dg) {
   if(dg == null || dg.DataSource == null) return null;
      BindingManagerBase bmb = dg
                           .BindingContext[dg.DataSource];
   if(bmb == null) return null;
   return bmb.Current;
}
```

Il suo scopo è quello di recuperare i dati "bindati" al DataGrid e non quelli realmente visualizzati. Come sappiamo infatti, è possibile applicare una sorta di template *(tableStyle)* alla griglia che, tra le altre cose, ci consente di nascondere alcune colonne. Il *BindingManagerBase* "legge" i dati realmente agganciati al DataGrid consentendo di recuperare anche le informazioni nascoste. Per usare correttamente il *BindingManagerBase*, attiviamo l'evento *CurrentCellChanged* del DataGrid accedendo alla sezione *eventi* del controllo ed implementiamo la seguente funzione:

```
private void dataGridClienti_CurrentCellChanged(
                        object sender, System.EventArgs e)
{
   System.Windows.Forms.DataGridCell selectedCell =
                              dataGridClienti.CurrentCell;
   object selectedItem = dataGridClienti[
                                selectedCell.RowNumber,
                              selectedCell.ColumnNumber];
   drvClienti = (DataRowView)
            GetCurrentBindedObject(dataGridClienti);
   lblClienteSelezionato.Text = drvClienti[
                          "Denominazione"].ToString();
   btnVisualizzadettaglio.Enabled = true;
}
```

In questo modo abbiamo la possibilità di accedere ad una *DataRowView* (una rappresentazione della riga selezionata) contenete tutti di dati associati.
Nel prossimo paragrafo vedremo come utilizzare la *DataRowView* per recuperare visualizzare i dati del cliente e generare la mappa.

INTEGRIAMO LE MAPPE

Aggiungiamo un nuovo form al progetto chiamandolo *DettaglioCliente.cs* ed aggiungiamo dei controlli. Nella prima parte inseriremo i dati relativi al cliente recuperati dalla *DataRowView* di cui abbiamo parlato in precedenza. Nella seconda parte inseriamo una PictureBox la cui immagine associata sarà la nostra mappa. Nell'evento *click* del bottone *"Genera Mappa"* definiamo una chiamata al nostro metodo *GetMap* visto in precedenza:

DETERMINAZIONE DEL PERCORSO

Anche in questo caso l'esempio è semplice e ben documentato:

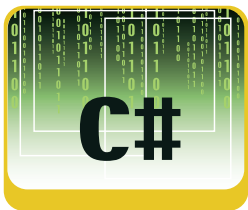
```
Sub UseDirectionsProperty()

   Dim objApp As New MapPoint
                        .Application
   Dim objMap As MapPoint.Map
   Dim objRoute As MapPoint.Route

   'Set up application
   Set objMap = objApp.ActiveMap
   Set objRoute = objMap.ActiveRoute
   objApp.Visible = True
   objApp.UserControl = True

   'Add route stops and calculate the route
   objRoute.Waypoints.Add
          objMap.FindResults("Seattle,
                        WA").Item(1)
   objRoute.Waypoints.Add
          objMap.FindResults("Redmond,
                        WA").Item(1)
   objRoute.Calculate

   MsgBox "The first direction is: " + _

objRoute.Directions.Item(1).Instruction

   End Sub
```

```
private void btnGeneraMappa_Click(object sender,
                               System.EventArgs e) {
   MapGenerator map = new MapGenerator();
   pictureBoxMappa.Image = map.GetMap(
      lblDenominazione.Text, lblIndirizzo.Text, lblCitta
         .Text, lblProvincia.Text, lblCap.Text, "Italy");
}
```

a tale metodo passeremo l'indirizzo completo del cliente da "mappare". Torniamo ora alla nostra classe *MapGenerator.cs* e completiamo l'implementazione del metodo *GetMap*. Come abbiamo detto in precedenza, per localizzare un punto su una mappa, a MapPoint basta un indirizzo. Modifichiamo quindi il metodo in modo da ricevere come parametri di input tutti i dati necessari a creare la mappa:

```
public Bitmap GetMap(string Denominazione, string
   Indirizzo, string Citta, string AltraCitta, string CAP,
                                    string Nazione)
{
   try{
      app = new ApplicationClass();
      app.Visible = false;
      map = new MapPoint.Map();
      map = app.ActiveMap;
```

Ricevuti questi parametri e creata l'istanza di *MapPoint*, il prossimo passo è quello di cercare l'indirizzo sulla mappa. Per *MapPoint* infatti, il dato da ricercare deve coincidere con il dati contenuti nel suo Data Base interno. Solo se l'indirizzo passato combacia con quello noto al software, il punto viene cor-

rettamente localizzato. Per eseguire la ricerca del punto sulla mappa si usa l'oggetto *FindResults* che non fa altro che ricercare tutti gli indirizzi noti sulla base di quello passato:

```
FindResults frs;
frs = map.FindAddressResults(Indirizzo, Citta,
        AltraCitta, string.Empty, CAP, Nazione);
IEnumerator ienum = frs.GetEnumerator();
ienum.MoveNext();
location = ienum.Current as MapPoint.Location;
```

il risultato del metodo *FindAddressResults* è una collection di *Location* (ricordate la location?) con tutti i dati recuperati. Per semplicità implementativa, consideriamo valido il primo punto della collection di indirizzi trovati quindi ci spostiamo sul punto individuato. Trovato il punto, non ci resta che evidenziarlo sulla mappa. Per farlo utilizziamo un altro semplice oggetto fornito da MapPoint: il *Pushpin*. Esso è semplicemente una sorta di "chiodino" utile ad indicare il punto sulla mappa.
La creazione di un oggetto *Pushpin* presuppone che esista una *Location* precisa individuata da MapPoint:

```
Pushpin PP;
if (location != null) {
    location.Select();
    location.GoTo();
    PP = map.AddPushpin(location, Denominazione);
```

Creato il *pushpin*, lo si può personalizzare come si vuole. Nel nostro caso specifico gli è stato assegnato un simbolo a forma di punes, è stato evidenziato ed è stato mostrato il Balloon con i dati del cliente:

```
    PP.Symbol = 4;
    PP.Highlight = true;
    PP.BalloonState =
            GeoBalloonState.geoDisplayBalloon;
}
```

Abbiamo quasi completato il lavoro. Se stessimo lavorando direttamente su MapPoint, a questo punto vedremmo la nostra mappa con il punto ben evidenziato e con tutti i dettagli del cliente. Ma all'inizio abbiamo detto che l'applicazione resta nascosta all'utilizzatore. Come fare allora? Si usa un semplice trucco: con il comando *CopyMap* che ha l'evidente scopo di effettuare una copia della mappa appena creata in memoria.
Una volta copiata la mappa, non ci resta che recuperarla dalla memoria e mostrarla a video passandola alla picture box che avevamo precedentemente inserito.

```
map.CopyMap();
```

```
    return BitmapFromClipboard();
    }catch (System.Runtime.InteropServices
                        .COMException comEx){
        MessageBox.Show("Si è verificato un errore
                durante la chiamata a MapPoint!!",
            "ioProgrammoMap", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
    }catch (Exception ex){
        MessageBox.Show("Si è verificato un errore
            nell\'applicazione!!", "ioProgrammoMap",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }finally{
        map.Saved = true;
        if(app != null)
            app.Quit();
            app = null;
    }
    return null;

}
```

In soccorso ci viene il metodo *BitmapFromClipboard* esposto di seguito:

```
private static Bitmap BitmapFromClipboard() {
    IDataObject iData = Clipboard.GetDataObject();
    if ((iData.GetDataPresent(DataFormats.Bitmap))) {
        Bitmap memoryImage = new Bitmap(((Bitmap)
            (iData.GetData("System.Drawing.Bitmap"))));
        return memoryImage;
    }
    return null;
}
```

L'immagine viene cosi rinviata al chiamante (il nostro form) e visualizzata all'utilizzatore della nosta applicazione.

CONCLUSIONI

Il software realizzato in queste pagine ci ha permesso di capire come interfacciare le nostre applicazioni ad un software molto interessante come MapPoint. Sebbene l'applicazione d'esempio sia semplice, il suo scopo è quello di stuzzicare la fantasia dei lettori. Provate ad immaginare quante applicazioni si possono realizzare! Quello che serve a MapPoint è un indirizzo, un luogo, un set di coordinate satellitari per poter disegnare una mappa precisa del luogo desiderato. L'integrazione con il pacchetto office ne consente un semplice utilizzo in abbinamento agli strumenti da esso forniti. La possibilità di includere MapPoint in applicazioni personalizzare lo rende ulteriormente flessibile.
Spazio alla fantasia allora! Gli strumenti ora ci sono!
Buon lavoro.

Michele Locuratolo

CONTATTA L'AUTORE

L'autore può essere contattato attraverso il suo blog su http://blogs.mindbox.it/mighell e sarà lieto di rispondere alle domande dei lettori.

PostgreSQL, un mostro di potenza

La versione 8.0 si apre a Windows rendendo il database più integrato con il sistema. Le funzionalità avanzate: Views, Stored Procedure, foreign keys lo rendono un DB eccezionale

Chi usa PHP è spesso abituato a farlo in congiunzione a MySQL. L'accoppiata è vincente a causa della leggerezza e della velocità di MySQL. Nonostante questo, MySQL nella versione attuale soffre di alcune limitazioni. Molto probabilmente queste limitazioni saranno superate nella versione 5 che attualmente è ancora in beta. Vero è che le versioni beta di MySQL spesso finiscono per diventare abbastanza stabili da poter essere usate in produzione, e tuttavia è altrettanto vero che una valida alternativa è costituita da PostgreSql che è dotato di tutte le caratteristiche di MySQL e anche di qualcuna in più. In questo articolo analizzeremo alcune interessanti funzioni di PostgreSQL e mostreremo come utilizzarle da PHP.

IL PRIMO DATABASE

Una prima annotazione importante da fare è che Postgres è multipiattaforma. Fino alla versione 7.4 però l'installazione in ambiente windows risultava piuttosto complessa.
Una delle innovazioni apportate dalla versione 8.0 è relativa alla disponibilità di un'installazione completamente grafica tipica dei sistemi Windows. Al termine dell'installazione vi troverete nel menu di Windows anche una voce relativa a PostgreSQL 8.0. All'interno di questo menu è disponibile il link per avviare *pgAdmin*, ovvero un'interfaccia completamente visuale che agevola gli utenti Windows nella normale amministrazione del sistema.
Viceversa in ambiente Linux , come di consueto e come in completa filosofia Unix si fa riferimento alla linea di comando.
In questo articolo utilizzeremo sempre la linea di comando per mostrare le caratteristiche essenziali del database. Tuttavia le operazioni possono essere facilmente replicate anche con *pgAdmin* in maniera del tutto visuale. Per la creazione del primo database sarà opportuno utilizzare l'utente *"postgres"* creato all'atto dell'installazione.
In ambiente Linux sarà sufficiente usare il comando *su postgres*, da *pgAdmin* verrà invece richiesta la password e l'utente per connettersi al db.
Per creare un database il comando da dare è

```
createdb mydb
```

dove *mydb* è il nome del nostro nuovo database, il server risponderà con un *CREATE DATABASE*. Da *pgAdmin* vedrete la nuova cartella con il database creato e una marea di altre cartelle fra cui spiccano *views/funzioni/funzioni trigger/tipi/operatori di classe*. Tutte cose che a chi ha un po' di conoscenza dell'architettura dei db più famosi risulteranno particolarmente familiari oltre che gradite.
Ovviamente potete sempre creare il database manualmente da riga di comando portandovi in *c:\programmi\postgresql \8.0\bin* e digitando

```
createdb -Upostgres mydb
```

vi verrà chiesto di inserire la password per l'utente postgres e il database verrà creato.

LA PRIMA TABELLA

Dop aver creato il primo database, il secondo passo sarà creare una tabella.
Lo faremo da riga di comando con

```
psql –Upostgres -s mydb
```

la risposta del server sarà

Benvenuto in psql 8.0.3, il terminale interattivo di PostgreSQL.

Digitare:

```
    \copyright per le condizioni di distribuzione
```

INSTALLARE POSTGRES
In ambiente windows potete utilizzare il file postgresql-8.0.msi, l'installazione procede con un tool grafico in pieno stile Microsoft Windows. Le uniche scelte in qualche modo importanti che potreste dover compiere sono relative al numero di estensioni da installare, questo dipende dal tipo di lavoro che dovete svolgere. In questo articolo non faremo uso di estensioni.

```
        \h per la guida sui comandi SQL
        \? per la guida sui comandi psql
        \g o terminare con punto e virgola per eseguire
                                              la query
        \q per uscire
mydb=#
```

per inserire i comandi abbiamo adesso due opportunità, scriverli in un file esterno e caricarli con *\i nomefile.ext* sostituendo a *nomefile.ext* il nome del file che contiene i comandi oppure digitarli manualmente nella console.
Al momento sceglieremo questa seconda strada tenendo ben presente che un'istruzione sarà eseguita solo in seguito all'immissione di un carattere di punto e virgola.
La nostra prima tabella sarà creata utilizzando i seguenti comandi:

```
mydb=# create SEQUENCE capsequence;
CREATE SEQUENCE

mydb=# CREATE TABLE Capitoli (
        mydb(# id_capitolo integer NOT NULL DEFAULT
                              nextval('capsequence'),
mydb(# PRIMARY KEY (id_capitolo)
mydb(# );
************(Modalità single step: comando di
                   verifica)*******************
CREATE TABLE Capitoli (
                 I  id_capitolo integer NOT NULL DEFAULT
                              nextval('capsequence'),
                              PRIMARY KEY (id_capitolo)
                                                     );
****(premere invio per procedere oppure digitare x
                     ed invio per annullare)****

NOTICE:  CREATE TABLE / PRIMARY KEY will create
                   implicit index "capitoli_pkey" f
or table "capitoli"
CREATE TABLE
mydb=#
```

Qui le cose sono leggermente diverse da come siamo abituati a vedere in MySQL. Per definire un campo autoincrementale abbiamo prima definito una "sequenza". Le squenze sono tabelle molto speciali dotate di una sola riga che contiene un puntatore che può essere manipolato tramite delle apposite funzioni.
Nel nostro caso nella definizione della tabella abbiamo utilizzato la funzione *"nextval"* per dire che il campo *id_capitolo* che è anche una *primary key* per la tabella deve contenere sempre il valore successivo a quello corrente della sequenza *"capsequence"*. In pratica inserendo un nuovo record nella tabella *"Capitoli"* verrà prima controllata la sequenza *"capsequence"*. Il campo *id_capitolo* verrà riempito con un valore incrementale e la sequenza verrà aggiornata con il nuovo valore.
Proviamo a vedere cosa succede dando il seguente comando due volte

```
mydb=# insert INTO capitoli VALUES(
                               nextval('capsequence'));
```

ed eseguendo subito dopo una *select*

```
mydb=# select * from capitoli;
```

il risultato è

id_capitolo

1
2
(2 righe)

ovvero esattamente quello che ci aspettavamo.
Potremmo provare qualcosa del genere

```
insert INTO capitoli VALUES(currval('capsequence'));
```

la funzione *currval* restituisce il valore attualmente contenuto nella sequenza *capsequence*. Il risultato di questa query questa volta sarebbe:

```
ERROR:  duplicate key violates unique constraint
                                         "capitoli_pkey"
```

È infatti impossibile assegnare due capitoli identici, poichè il campo *id_capitolo* è definito come *primary key*. L'ultima funzione utile rispetto alle sequenze è la *setval* che consente di variare i parametri della sequenza.
Ad esempio

```
select setval('capsequence',18);
```

fa in modo che *nextval* restituisca 19, oppure

```
select setval('capsequence',18,false);
```

fa in modo che *nextval* restituisca 18.
Si intuisce facilmente che si possono manipolare campi assegnandoli a sequenze diverse.

MODIFICARE UNA TABELLA

Prima di tutto, creiamo una seconda tabella e poi una terza. Stiamo cercando di creare un database di libri. Avremo bisogno di una tabella di autori, ciascun autore avrà scritto dei libri, ciascun libro sarà diviso in capitoli.

La tabella degli autori sarà la seguente

```
create table autori (
id_autore integer NOT NULL DEFAULT nextval('s_autori'),
nome text,
cognome text,
primary key(id_autore)
)
```

La tabella dei libri sarà invece la seguente

```
create table libri (
id_libro integer NOT NULL DEFAULT nextval('s_libri'),
id_autore integer references autori,
titolo varchar(80),
primary key(id_libro)

)
```

In questa seconda tabella notiamo la presenza di un'istruzione *"references autori"* nella seconda riga. Questa istruzione setta una *"foreign key"*, ovvero un'istruzione di integrità referenziale. Nessun inserimento potrà essere fatto nella tabella libri se nel campo *id_autore* non sarà presente un valore già inserito nella tabella autori.
Proviamo a fare un test per vedere se quanto abbiamo detto è corretto:

```
insert INTO libri (id_autore,titolo) VALUES
                                   ('8','Imparare PHP');
```

restituisce

```
ERROR:  insert or update on table "libri" violates
        foreign key constraint "libri _id_autore_fkey"
DETAIL:  Key (id_autore)=(8) is not present in table
                                              "autori".
```

Proviamo a inserire l'autore corretto nella tabella degli autori ed eseguiamo di nuovo il test:

```
insert into autori (nome,cognome) VALUES
                                     ('Fabio','Farnesi');
INSERT 18408 1
insert INTO libri (id_autore,titolo) VALUES
                                   ('1','Imparare PHP');
INSERT 18409 1
```

Questa volta tutto è andato a buon fine. Proviamo a cancellare l'autore dalla tabella degli autori

```
delete from autori where id_autore=1;
```

il risultato questa volta è:

```
ERROR:  update or delete on "autori" violates foreign
        key constraint "libri_id_autore_fkey" on "libri"
DETAIL:  Key (id_autore)=(1) is still referenced from
                                         table "libri".
```

In poche parole non è possibile cancellare il record dalla tabella autori perché ci sono record nella tabella libri che lo referenziano.
Per potere manipolare tabelle che hanno dei campi dichiarati come *foreign keys* è necessario ridefinire le azioni *ON UPDATE* e *ON DELETE*, ovvero dobbiamo programmare PostgreSQL per reagire in maniera opportuna ad un evento di *UPDATE* o *DELETE* di una chiave referenziata. Per farlo modifichiamo la tabella libri come segue:

```
create table libri (
id_libro integer NOT NULL DEFAULT nextval('s_libri'),
id_autore integer references autori ON UPDATE
                             CASCADE ON DELETE CASCADE,
titolo varchar(80),
primary key(id_libro)
);
```

Questa volta eseguendo

```
delete from autori where id_autore=1;
```

non solo verrebbe cancellato il record nella tabella autori ma anche tutti i record nella tabella libri che contengono un riferimento ad un autore il cui id è uguale a 1. Questo modo di procedere non è l'unico possibile, si potrebbe anche utilizzare il seguente metodo:

```
create table libri (
id_libro integer NOT NULL DEFAULT nextval('s_libri'),
id_autore integer references autori ON UPDATE
                             CASCADE ON DELETE SET NULL,
titolo varchar(80)
);
```

Così facendo in seguito alla cancellazione di un record nella tabella autori si otterrebbe che tutti i campi *id_autore* corrispondenti nella tabella libri venissero settati a *NULL*. Nessun dato verrebbe cancellato, ma verrebbe semplicemente spezzata la relazione con la tabella autori. Per finire ridefiniamo la tabella capitoli come segue:

```
create table capitoli (
id_capitolo integer NOT NULL DEFAULT
                                 nextval('capsequence'),
id_libro integer references libri ON UPDATE CASCADE
                                     ON DELETE CASCADE,
id_autore integer references autori ON UPDATE
                             CASCADE ON DELETE CASCADE,
introduzione text,
body text
);
```

Utilizza questo spazio per le tue annotazioni

EREDITARIETÀ DELLE TABELLE

Per quanto le *foreign keys* siano interessanti non sono una conquista di PostgreSQL, MySQL le usa sulle tabelle innodb già da qualche tempo, tuttavia MySQL parserizza correttamente l'istruzione *references* ma non la immagazzina nella tabella così che un eventuale dump della tabella risulterebbe privo delle *foreign keys*, viceversa PostgreSQL esegue correttamente questa funzione.
Nonostante questo, si tratta ancora di differenze marginali. Una prima differenza importante è data dall'esistenza della clausola *INHERITS*.
Supponiamo ad esempio di volere conoscere tutti gli autori di libri che fanno anche parte della redazione di una testata giornalistica.
L'elegante soluzione proposta da PostgreSQL è la seguente

```
create table redattori (
testata char(40)
) INHERITS (autori);
```

Una qualunque query sulla tabella autori verrà adesso estesa anche alla tabella redattori, a meno che non sia diversamente specificato, ad esempio:

```
insert into autori (nome,cognome) VALUES
                                   ('Fabio','Farnesi');
insert into redattori (nome,cognome,testata) VALUES
                    ('Thomas','Zaffino','Internet Magazine');
```

eseguendo una select di questo genere:

```
select * from autori;
```

mi viene restituito

```
id_autore | nome    | cognome
-----------+----------+---------
    2      | Fabio   | Farnesi
    3      | Thomas | Zaffino
```

anche se in realtà il secondo insert era stato eseguito sulla tabella redattori e non sulla tabella autori.
Se adesso volessi conoscere gli autori che non sono anche redattori potrei eseguire:

```
select * from ONLY autori;
```

restituisce

```
id_autore | nome    | cognome
-----------+----------+-----------
    2      | Fabio   | Farnesi
(1 riga)
```

eseguendo la query sulla sola tabella autori. Viceversa per conoscere gli autori che sono anche redattori

```
select * from redattori;
```

restituisce

```
id_autore |  nome   | cognome | testata
-----------+---------+-----------+-----------------------
     4     | Thomas |  Zaffino  | Internet Magazine
(1 riga)
```

Si tratta di una funzione decisamente interessante.

VISTE E STORED PROCEDURE

Anche in questo caso si tratta di funzionalità che verranno implementate solo in MySQL 5.0.
Supponiamo di avere una query mediamente complessa e ripetitiva, ad esempio:

```
select nome,cognome,titolo,introduzione from
                              autori,libri,capitoli where
autori.id_autore=capitoli.id_autore;
```

in PostgreSQL è valida un'istruzione del genere:

```
create VIEW autori_capitoli AS
select nome,cognome,titolo,introduzione
from autori,libri,capitoli
where autori.id_autore=capitoli.id_autore;
```

una *view* contiene esattamente il risultato della query che la definisce. Su una *view* è lecita un'istruzione del genere:

```
select * from autori_capitoli;
```

che restituirebbe esattamente il contenuto della *view.* In realtà le view vengono implementate tramite il *rule system.* In sostanza la creazione di una view di questo tipo:

```
CREATE VIEW myview AS
SELECT * FROM myTable;
```

viene espansa dal *rule system* come segue:

```
CREATE TABLE myview (con le stesse colonne della
                                    tabella myTable);
CREATE RULE "_RETURN" AS ON SELECT TO myview
                                          do INSTEAD
            SELECT * FROM myTable;
```

La sintassi è facilmente comprensibile, si tratta di una regola che avverte Postgres che ogni *SELECT* verso la tabella *myview* deve essere ese-

guita eseguendo una *Select* su *myTable;* in pratica una *'SELECT * FROM myview where condizione'* verrebbe espansa in:

```
select * from (
select * from mytable) as nome_subquery where
                                   condizione;
```

e con questo abbiamo anche introdotto il concetto di subquery che è ampiamente supportato in PostgreSQL, mentre è supportato con qualche limitazione solo da MySQL 4.1 e superiori.
Va da se che se questa è la logica con cui lavora il '*system rules*' è possibile anche creare delle regole per l'inserimento, l'update e la cancellazione dei dati in una tabella. Considerate il seguente esempio:

```
create table log_libri(
id_log integer NOT NULL DEFAULT nextval(
                              'logsequence'),
libro_changed text,
libro_current text,
log_who text,
log_when timestamp with time zone default now()
);
```

Si tratta di una tabella in cui vorremmo inserire il log dei vari *UPDATE* eseguiti sulla tabella libri. Grazie al *'System rules',* è possibile creare una regola del genere:

```
create rule librilogger AS on UPDATE TO libri
WHERE NEW.titolo <> OLD.titolo
DO INSERT INTO log_libri
VALUES(nextval('logsequence'),
NEW.titolo,OLD.titolo,current_user,
CURRENT_TIMESTAMP);
```

Tale che ogni volt ache si scatena un evento di tipo *UPDATE* sulla tabella libri se la colonna titolo resulta cambiata venga inserita una riga nella corrispondente tabella dei log.
Ed infatti eseguendo:

```
update libri set titolo='Programmare con i Database'
                       where titolo = 'Imparare SQL';
```

e di seguito

```
select * from log_libri;
```

ottengo

id_log| libro_changed |libro_current | log_who | log_when
------+--------------+-------------+--------+----------
1 |Programmare con i Database| Imparare SQL | postgres | 2005-06-24 10: 29:21.937+02

USIAMOLO DA PHP

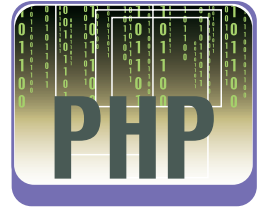
Il modulo da abilitare in PHP per poter utilizzare PostgreSQL è *pgsql,* potete farlo al solito dal php.ini decommentando *pgsql.dll* o *pgsql.so* nella riga delle estensioni , a seconda che usiate Windows o Linux. Tutte le funzioni relative a Postgres iniziano con *pg_,* pertanto uno script per la connessione al database potrebbe essere il seguente:

```
<?
$dbconn = pg_connect("host=localhost port=5432
                     dbname=mydb user=ioprogrammo");
?>
```

A questo punto potremmo introdurre un discorso sul sofisticato sistema di autenticazione e di gestione dei permessi utilizzato da PostgreSQL. Non riusciamo a farlo per questioni di spazio, è utile però sapere che postgres riesce a controllare diversi metodi di autenticazione e che la gestione dei permessi arriva fino alla singola colonna. Le altre funzioni utilizzabili da PHP per quanto riguarda PostgreSQL sono molto simili a quelle usate da MySQL.
Ad esempio una query può essere effettuata come segue:

```
<?
$dbconn = pg_connect("host=localhost port=5432
                     dbname=mydb user=ioprogrammo");
if (!$result) {
   echo "errore.\n";
   exit;
}
while ($row = pg_fetch_row($result, $i)) {
  for ($j=0; $j < count($row); $j++) {
         echo Åg$row[$j]&bnsp;
  }
}
?>
```

CONCLUSIONI

PostgreSQL è un database stracolmo di funzionalità, da questo articolo rimangono fuori i trigger, le funzioni, la conversione dei tipi, le estensioni di PostgreSQL, la sicurezza, e molto altro ancora. Speriamo comunque di avere messo in mostra una parte delle funzionalità che rendono PostgreSQL un'alternativa molto comoda a MySQL, specialmente con la versione 8.0 che abbatte e rende semplice l'installazione in ambiente Windows, pertanto aumenta l'usabilità di questo.

ASP.NET 2.0, meno codice per tutti

Il framework .NET 2.0 è ormai alle porte, con tutte le novità che ne conseguiranno per i programmatori e in particolare per quelli web. Vediamo cosa dovremo aspettarci dal nuovo Framework

Al momento in cui scrivo questo articolo è stata rilasciata la versione beta 2 di Visual Studio .NET 2005 e del .NET Framework 2.0. Dalla prima beta ad oggi sono cambiate diverse cose: alcune classi sono state rimosse, alcuni nuovi controlli sono stati messi in cantiere, ma la filosofia di base del nuovo Framework e le evoluzioni più importanti sono rimaste intatte. È proprio di queste che parleremo nel corso di questo articolo.

FINO AL 70% IN MENO DI CODICE

La novità più rilevante riguarda la possibilità di scrivere applicazioni usando molto codice in meno rispetto al recente passato. In alcuni casi si raggiunge un risparmio addirittura del 70%. Questa innovazione si basa sul raggruppamento di istruzioni ripetitive all'interno di nuovi controlli. Prendiamo ad esempio una funzionalità molto utilizzata: la richiesta di dati ad un database e la successiva visualizzazione all'interno di una griglia. In ASP.NET 1.1 i passi da seguire erano questi:

1. Dichiarazione, inizializzazione ed utilizzo di un oggetto *Connection*.
2. Dichiarazione, inizializzazione ed utilizzo di un oggetto *DataAdapter*.
3. Dichiarazione e riempimento di un *DataSet*.
4. Utilizzo dell'evento *Load* della pagina per impostare il *DataSource* della griglia in modo da visualizzare i dati.

In totale circa una trentina di istruzioni da scrivere all'interno del codice, ma soprattutto da riscrivere nel caso in cui un'altra pagina avesse dovuto effettuare un'altra query al database. Ecco, invece, come fare per ricavare i dati utilizzando il nuovo controllo *SqlDataSource*:

```
<asp:SqlDataSource
ID="SqlDataSource1"
runat="server"
ConnectionString="<%$ ConnectionStrings:
                    NorthwindConnectionString %>"
SelectCommand="SELECT * FROM [Category Sales
                                   for 1997]">
</asp:SqlDataSource>
<asp:GridView ID="GridView1"
runat="server"
AutoGenerateColumns="True"
DataSourceID="SqlDataSource1">
</asp:GridView>
```

Il nuovo controllo *SqlDataSource* contiene due proprietà, la *ConnectionString* per specificare la stringa di connessione verso un database SQL Server, e la *SelectCommand* per specificare l'istruzione SQL da eseguire per ricavare i dati dal database. Grazie a questo nuovo controllo, la *GridView* può visualizzare il risultato della query senza dover scrivere nessun altro tipo di informazioni! Sarà il controllo stesso ad aprire e chiudere la connessione, creare un oggetto *DataAdapter,* eseguire la query e restituire i dati. Annotiamo due cose interessanti riguardo questo semplice esempio; la prima è che la stringa di connessione viene ricavata direttamente dal file *web.config* con la nuova sintassi *$ sezione:chiave,* la seconda cosa è l'utilizzo del nuovo controllo *GridView* che rimpiazza e potenzia il vecchio controllo *DataGrid.*

NUOVA AMMINISTRAZIONE DEI SITI

L'amministrazione dei siti ASP.NET era gestita tramite due file XML: *machine.config* e *web.con-*

fig. Nel primo venivano impostati i parametri relativi a tutto il server su cui erano in esecuzione i siti web mentre il secondo serviva per sovrascrivere o aggiungere dei settaggi specifici per un sito ASP.NET. Anche se i file XML sono leggibili e chiari non sono certo paragonabili ad un'interfaccia Windows tipo MMC o simili. Ogni nuovo sito ASP.NET avrà una sezione di configurazione e di amministrazione visuale (vedi **Figura 1**).

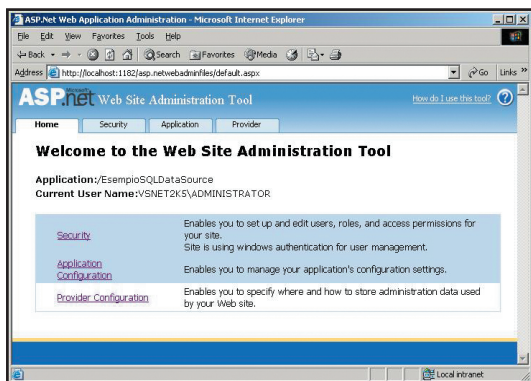



***Fig. 1:** La classica interfaccia visuale di Amministrazione di un sito in tecnologia Asp 2.0*

Il sito di amministrazione sarà accessibile tramite web, per cui sarà possibile anche un'amministrazione da remoto. Il programma è suddiviso in tre sezioni:

1. **Security:** Tramite questa sezione è possibile gestire tutto ciò che è inerente alla sicurezza del proprio sito. Si possono aggiungere utenti, ruoli e creare regole di accesso al sito (ad esempio, alcuni ruoli possono vedere pagine che altri non possono vedere, ecc.).

2. **Application Configuration:** Tramite questa sezione è possibile gestire le impostazioni dell'applicazione web. Aggiungere, rimuovere e modificare voci all'interno del file *web.config* diventerà molto intuitivo e visuale.

3. **Provider Configuration:** Permette di configurare il database dove memorizzare le impostazioni di amministrazione del proprio sito web.

ASP.NET 2.0 offre una serie di nuovi controlli come il *Login*, il *LoginStatus*, il *CreateUserWizard*, e molti altri, che si interfacciano automaticamente con il database di amministrazione permettendo di autenticare un utente e ricavarne il ruolo, di creare un nuovo utente con alcune opzioni come la generazione di una password e l'invio di email per la conferma di iscrizione, ecc. il tutto scrivendo pochissimo codice!

MASTER PAGES

Le *Master Pages* sono una vera e propria innovazione inserita all'interno di questa nuova versione di ASP.NET. Una pagina che viene dichiarata *Master* detterà il formato grafico di tutte quelle pagine che dichiareranno di appartenere ad una pagina *master*. Il classico layout grafico del portale Internet, ad esempio, formato da un'intestazione, da un piè di pagina, un menu laterale e il contenuto centrale, può essere realizzato molto facilmente adoperando una *master page* per tutto ciò che rimane fisso (intestazione, piè di pagina, ecc.) e una serie di pagine di contenuto, chiamate anche *Content Pages*, che dichiarano di utilizzarla (vedi **Figura 2**).

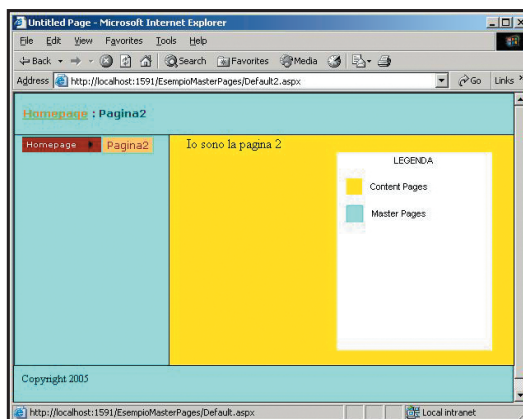



***Fig. 2:** Le pagine dichiarate appartenenti a una pagina master ne ereditano l'interfaccia di base*

La sezione in giallo è quella che, in base alla selezione del menu di sinistra, cambia il proprio contenuto. La sezione in azzurro rappresenta tutto ciò che è stato inserito all'interno della *Master Pages* e che definisce il layout della pagina ASP .NET.
Questo semplice esempio utilizza due nuovi controlli aggiunti in questa versione di ASP .NET: il *SiteMapPath* e il *Menu*. Entrambi i nuovi controlli possono basarsi su un nuovo file XML chiamato *web.sitemap* contenente la struttura gerarchica delle pagine all'interno del proprio sito. Ecco un esempio:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com
                          /AspNet/SiteMap-File-1.0" >
   <siteMapNode url="~/Default.aspx" title=
                                    "Homepage">
      <siteMapNode url="~/Default2.aspx"
                               title="Pagina2" />
   </siteMapNode>
</siteMap>
```

Il controllo *SiteMapPath* utilizza questo file per

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

visualizzare la posizione gerarchica della pagina correntemente visualizzata dando la possibilità di navigare tra le pagine ad essa relazionate. Il controllo *Menu* utilizza anch'esso il file web.sitemap per mostrare la gerarchia delle pagine e permettere la navigazione all'interno del sito. Inoltre, il controllo *Menu,* fornisce effetti a tendina e dinamica tra le sue voci, senza scrivere una riga di codice!

TEMI E PERSONALIZZAZIONE

L'aspetto grafico di un sito può dipendere dalla definizione di stili, come colori e font, all'interno di un file *CSS*. ASP.NET 2.0 va oltre, prevedendo la possibilità di applicare temi al proprio sito anche a run-time! Realizzando vari modelli grafici per il proprio sito sarà possibile far scegliere all'utente quello a lui più gradevole e cambiarlo in tempo reale. Inoltre, grazie ad una serie di nuove classi create per la personalizzazione, sarà possibile associare, tra l'altro, questa scelta all'utente in modo che successivamente il sito si modelli in base alle sue preferenze.
Come abbiamo visto nella sezione dedicata all'amministrazione del sito, è possibile definire gli utenti ed i gruppi che possono accedere all'applicazione web. Questi dati vengono utilizzati da una serie di nuovi controlli Login che permettono di implementare, molto facilmente, una pagina di autenticazione e di registrazione di utenti. Una volta autenticato un utente, il sito conoscerà tutte le sue eventuali preferenze, grafiche e di contenuto, permettendogli di adattarsi automaticamente. I temi si basano sul nuovo concetto di skin ereditato da software famosi come Windows Media Player o Winamp. Tramite una skin è possibile definire l'aspetto di un singolo controllo all'interno della pagina. Ad esempio, il seguente skin definisce un font di dimensione media per una textbox ed una label:

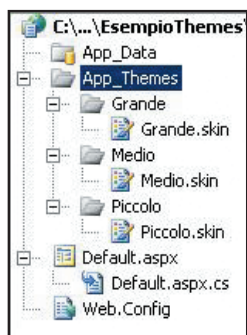



Fig. 3: ***Un esempio di gerarchia di directory relativa a tre temi***

```
<asp:Label Font-Size="Medium" RunAt="Server" />
<asp:TextBox Font-Size="Medium" RunAt="Server" />
```

Le informazioni dovranno essere messe all'interno di un file che abbia estensione *.skin* e inserite all'interno di una sotto directory all'interno della directory speciale *App_Themes*.
Il nome della sotto directory è molto importante in quanto verrà utilizzata da ASP.NET 2.0 per capire quale tema applicare al sito. Ad esempio in **Figura 3** è rappresentata la gerarchia di directory necessaria a definire tre temi. Mentre in **Figura 4** è rappresentata l'applicazione che cambia dinamicamente l'aspetto grafico della pagina ASP.NET. Selezionando un valore dalla combo box le dimensioni delle label e dei campi di testo cambieranno dinamicamente.

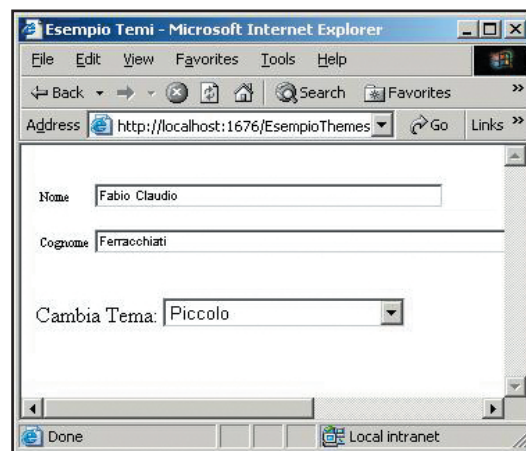



Fig. 4: ***È possibile selezionare un tema semplicemente dalla ComboBox***

Le righe di codice per scrivere questo esempio sono ben tre:

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Page.Theme = Request["DropDownList1"];
}
```

Il nuovo evento PreInit viene evocato prima di effettuare l'inizializzazione della pagina permettendoci di variare il tema della pagina.

CONCLUSIONI

La nostra carrellata sulle novità di ASP.NET 2.0 finisce qui, per ora. L'innovazione maggiore apportata dalla versione 2.0 del framework è relativa all'integrazione di molte righe di codice in singoli controlli. Questo diminuisce il tempo di scrittura del codice ed aggiunge un nuovo livello di visibilità al sistema, che nasconde ancora di più al programmatore quanto avviene nelle parti sottostanti, pur permanendo le possibilità di personalizzazione. Non si tratta certo di innovazioni di poco conto!

Fabio Claudio Ferracchiati

Hacking IIS con ASP.NET e i moduli

Hacking non sempre vuol dire intrusione, in questo caso modificheremo il cuore di IIS riscrivendo completamente la logica di gestione delle pagine, ridefinendone il comportamento standard

Vi siete mai chiesti cosa succede quando richiamate una pagina ASP.NET? A servirla generalmente è un server IIS. Proviamo in linea del tutto generale a capire cosa succede. La richiesta arriva al server, il server controlla l'estensione della pagina richiesta ed avvia una serie di procedure. Nel caso in cui la pagina richiesta sia una pagina gestibile dal framework .NET, le richieste vengono inviate ad un'istanza di *httpruntime*. Questa istanza usa un *httpapplicationfactory* per creare un oggetto di tipo *httpapplication* che processa la richiesta. L'*Httpapplication* mantiene una collezione di *httpmodules*. I moduli sono una sorta di filtro che viene eseguito prima che la richiesta sia soddisfatta dal gestore appropriato. Ad esempio è un modulo a gestire lo stato della sessione. Quando i moduli hanno esaurito il loro compito passano il controllo al gestore, ovvero l'handler che si occupa di parserizzare la pagina e restituire l'output corretto. Dopo che il gestore ha eseguito il suo compito il controllo torna di nuovo al modulo, per cui la sequenza corretta è la seguente:

```
1 - http module [Begin Request]
2 - http handler
3 - http module [End Request]
```

Ora, quanto abbiamo detto è estremamente semplificato, tuttavia ci serve per capire alcune cose fondamentali:

1) Le pagine possono essere gestite da un handler personalizzato.
2) Possiamo creare nuovi moduli da eseguire prima di passare il controllo all'handler.
3) Possiamo fare in modo che particolari estensioni siano gestite da un handler specifico.

In pratica possiamo manomettere il normale percorso della pipeline di gestione di una pagina creando noi dei percorsi adeguati e sostituendo quello di default. Creeremo dei moduli che verranno eseguiti prima del gestore e collezioneranno una serie di informazioni che riutilizzeremo a scopo statistico.

Impegno

Tempo di realizzazione

HTTPHANDLERS E HTTPMODULES: COSA SONO

Per definizione, un *HttpHandler* è un componente .NET che implementa l'interfaccia *System.Web .IHttpHandler.* Allo stesso modo, un *HttpModule* implementa, invece, l'interfaccia *System.Web.IHttpModule.* Un *HttpHandler* è, quindi, un filtro in grado di gestire tutte le richieste che pervengono verso una determinata risorsa rappresentando, difatti, l'*entry-point* di quella risorsa. Anche ASP.NET fa ampio uso di *HttpHandlers* e *HttpModules,* tutti definiti nel *machine.config:.* Gli *HttpHandlers,* come gli *HttpModules,* intervengono nella pipeline ASP.NET come rappresentato in **Figura 1**.

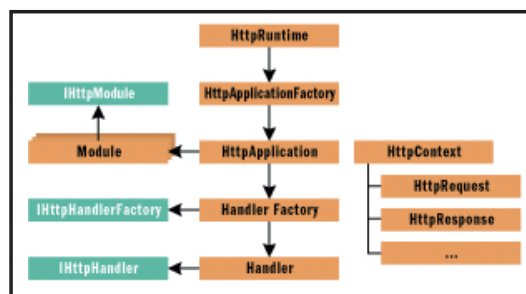



Fig. 1: *La pipeline di ASP.NET.*

Nell'immagine è possibile notare come una richiesta attraversa diverse fasi prima di giungere al gestore, che infine la elabora e restituisce la pagina web vera e propria. Nel processo intervengono i diversi moduli impostati a livello di applicazione o di macchina, ed infine il gestore designato per la risorsa richiesta. Ne deduciamo che i moduli possono essere più di uno e validi per tutte le risorse dell'applicazione, mentre i gestori sono specifici per singola risorsa.

IL PROGETTO MYSTATS

Per comprendere il funzionamento dei moduli e dei gestori http, ci occuperemo dello sviluppo di un'applicazione per la raccolta di informazioni statistiche sul nostro sito. Un modulo http intercetterà tutte le richieste indirizzate verso le pagine aspx. Successivamente un gestore http sarà il responsabile della generazione di un report rappresentante i dati raccolti. Nel dettaglio l'applicazione terrà traccia delle informazioni di seguito riportate:

- Data ed ora di visita;
- Pagina visitata;
- Tempo di esecuzione;
- Indirizzo IP del client;
- Browser utilizzato;

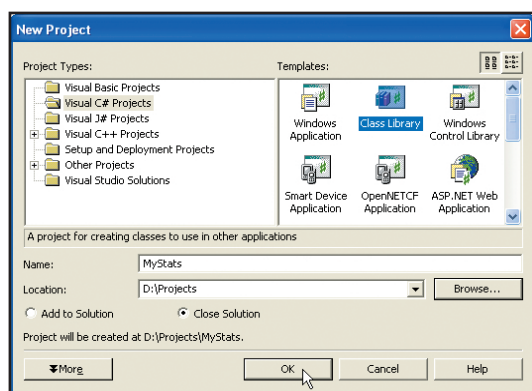



***Fig. 2:* La creazione di un nuovo progetto ASP.NET.**

- Sistema operativo in uso;
- Pagina di provenienza;

I dati raccolti saranno memorizzati in una collection e mantenuti nell'oggetto cache ASP.NET, che conterrà, come impostazione di default, le statistiche relative agli ultimi venti accessi. Per creare il progetto *MyStats,* apriamo Visual Studio .NET e creiamo un nuovo progetto *Class Library* in C# come visualizzato in **Figura 2**. Aggiungiamo un riferimento all'*assembly System .Web.dll* cliccando sul menu *Progetto -> Aggiungi Riferimento.* A questo punto possiamo creare la seguente struttura:

```
publicstructStatManager
{ public DateTime DataVisita;
  public string URLPagina;
  public TimeSpan TempoDiEsecuzione;
  public string IP;
  public string Browser;
  public string SistemaOperativo;
  public string URLReferer; }
```

Questa struttura è il contenitore designato per gestire le informazioni relative ad ogni singola visita. Come anticipato, sfruttando le potenzialità dell'oggetto cache di ASP.NET, possiamo mantenere in memoria un array di strutture di tipo *StatManager* senza occuparci, per ora, di una memorizzazione fisica su database.

IL MODULO STATMODULE

Il modulo *StatModule* raccoglierà le informazioni relative ad ogni singola visita, gestendo la storicizzazione nell'oggetto cache. Al nostro progetto aggiungiamo una nuova classe e la chiamiamo *StatModule.cs.* Nella dichiarazione della classe aggiungiamo l'interfaccia *IHttpModule*:

```
public classStatModule : IHttpModule
```

L'implementazione dell'interfaccia *IHttpModule* è l'esplicita dichiarazione della classe come modulo http, costituendo, difatti, il punto di ingresso del runtime asp.net. L'interfaccia comporta l'implementazione dei metodo *Init* e *Dispose.* Il primo accetta come parametro l'oggetto *HttpApplication,* contenitore del contesto della chiamata e che consente l'interazione con gli eventi propri dell'applicazione. Inseriamo, quindi, il codice che consente l'attivazione dei gestori degli eventi *BeginRequest* e *EndRequest,* come di seguito riportato:

```
publicvoidInit(HttpApplication context)
{ context.BeginRequest += new EventHandler(
                              context._BeginRequest);
  context.EndRequest += new EventHandler(
                              context._EndRequest); }
```

Il metodo *Init* imposta i gestori per gli eventi *BeginRequest,* il primo evento generato in risposta ad una richiesta inoltrata ad ASP.NET, ed *EndRequest,* generato, invece, come ultimo evento nella pipeline ASP.NET. Una delle caratteristiche della nostra applicazione è quella di calcolare il tempo impiegato per l'esecuzione della pagina. Per cui creiamo una variabile a livello di classe che, nell'evento *BeginRequest,* si occupa di memorizzare la data e l'ora di inizio della richiesta:

```
privateDateTime _startDate;
...
privatevoidcontext_BeginRequest(object sender,
                                      EventArgs e)
{ _startDate = DateTime.Now; }
```

Lo stato del modulo http viene mantenuto per tutto il ciclo di vita della stessa richiesta, pertanto nel gestore dell'evento *EndRequest,* fulcro della nostra applicazione, non avremo alcuna difficoltà a calcolare il tempo di esecuzione. Nello stesso metodo possiamo, quindi, recuperare tutte le informazioni statistiche per poi raccoglierle in un oggetto di tipo *StatManager.* Il codice mostrato di seguito evidenzia

Utilizza questo spazio per le tue annotazioni

• PIPELINE: è la "condutura" attraverso la quale passano le richieste web;
• OGGETTO CACHE: fornisce un'infrastruttura potente per la memorizzazione di dati temporanei. Viene creata un'istanza per ogni applicazione;
• ESTENSIONI ISAPI (INTERNET SERVICE API): Un componente C++ caricato dal web server ed utilizzato per estenderne le funzionalità.

come raccogliere e storicizzare le informazioni all'interno del gestore dell'evento *EndRequest:*

```
privatevoidcontext_EndRequest(object sender,
                                        EventArgs e)
{ HttpApplication context = (HttpApplication)sender;
   string path = context.Context.Request.Path
                                        .ToLower();
   if (path.EndsWith(".axd")) return;
   StatManager sm = new StatManager();
   sm.TempoDiEsecuzione =
               DateTime.Now.Subtract(_startDate);
   sm.URLPagina = context.Request.Url.ToString();
   sm.URLReferer = context.Request.ServerVariables[
                                  "HTTP_REFERER"];
   sm.Browser = context.Request.Browser.Type;
   sm.SistemaOperativo =
                  context.Request.Browser.Platform;
   sm.DataVisita = DateTime.Today;
   sm.IP = context.Request.UserHostName;
   ArrayList list;
   if (context.Context.Cache.Get("MyStats")!=null)
     list = (ArrayList)context.Context.Cache["MyStats"];
   else
     list = new ArrayList();
   if (list.Count==20) list.RemoveAt(0);
     list.Add(sm);
   context.Context.Cache["MyStats"] = list; }
```

Il codice evita di creare statistiche per i file *".axd"* con l'inserimento di un semplice controllo. Inoltre, solo a scopo dimostrativo, viene impostato un limite massimo di 20 registrazioni statistiche contemporanee, un vincolo superabile nel caso in cui si decidesse di utilizzare un database.

SUL WEB

Microsoft ASP.NET
http://msdn.microsoft.com/asp.net

ASPItalia.com
http://www.aspitalia.com

Il mio blog
http://blogs.ugidotnet.org/fabioc

IL PROCESSO DI RICHIESTA DI ASP.NET

Il processo di richiesta di ASP.NET è basato su una pipeline, una "conduttura" attraverso la quale la richiesta, partita dal client e filtrata dal web server, arriva fino al gestore http designato. Ogni richiesta, attraversa zero o più moduli http, condivisi a livello di applicazione o a livello di macchina, per essere elaborata. Un modulo, infatti, ha il controllo completo della richiesta e risponde ad una determinata sequenza di eventi di richiesta o eventi di applicazione: *BeginRequest, AuthenticateRequest, AuthorizeRequest, ResolveRequestCache, AcquireRequestState* e *PreRequestHandlerExecute.* Successivamente, viene richiamato il gestore della risorsa richiesta. Infine, la risposta ripercorre la pipeline dal gestore attraverso i moduli http, che implementano gli eventi di risposta o di applicazione: *PostRequestHandlerExecute, ReleaseRequestState, UpdateRequestCache* e *EndRequest.*

IL GESTORE STATHANDLER

Dopo aver memorizzato le statistiche nella cache, creiamo ora una semplice pagina per poterle visualizzare sfruttando le potenzialità dell'*HttpHandler*

ioProgrammo - MyStats

Data	Pagina	IP	Browser	Tempo(ms)	Referer	S.O.
12/04/2005	scheda.aspx	127.0.0.1	IE6	421	prodotti.aspx	WinXP
12/04/2005	default.aspx	127.0.0.1	IE6	265	scheda.aspx	WinXP
12/04/2005	prodotti.aspx	127.0.0.1	IE6	265	default.aspx	WinXP
12/04/2005	scheda.aspx	127.0.0.1	IE6	0	prodotti.aspx	WinXP
12/04/2005	default.aspx	127.0.0.1	IE6	0	scheda.aspx	WinXP
12/04/2005	prodotti.aspx	127.0.0.1	Netscape5	0	default.aspx	WinXP
12/04/2005	scheda.aspx	127.0.0.1	Netscape5	0	prodotti.aspx	WinXP
12/04/2005	default.aspx	127.0.0.1	Netscape5	0	scheda.aspx	WinXP
12/04/2005	prodotti.aspx	127.0.0.1	IE6	0	default.aspx	WinXP
12/04/2005	prodotti.aspx	127.0.0.1	Netscape5	0	default.aspx	WinXP

Fig. 3: Il risultato delle statistiche

fino ad ottenere un output simile a quello visualizzato in **Figura 3**. Aggiungiamo quindi una nuova classe al progetto, la chiamiamo *StatHandler* ed impostiamo l'interfaccia *IHttpHandler* per la classe:

```
publicclassStatHandler : IHttpHandler
```

Implementiamo ora i metodi dell'interfaccia *IsReusablee ProcessRequest:*

```
 public bool IsReusable
 { get{return true;} }
 public void ProcessRequest(HttpContext context)
 { ArrayList list = (ArrayList)context.Cache["MyStats"];
     context.Response.Write("<h1>ioProgrammo -
                                MyStats</h1>");
     context.Response.Write("<table border=\"1\">");
     context.Response.Write("<tr bgcolor=
                                \"#eeeeee\">");
     context.Response.Write("<td valign=
                        \"top\"><b>Data</b></td>");
     context.Response.Write("<td valign=
                      \"top\"><b>Pagina</b></td>");
     context.Response.Write("<td valign=
                          \"top\"><b>IP</b></td>");
     context.Response.Write("<td valign=
                     \"top\"><b>Browser</b></td>");
     context.Response.Write("<td valign=
                   \"top\"><b>Tempo(ms)</b></td>");
     context.Response.Write("<td valign=
                     \"top\"><b>Referer</b></td>");
     context.Response.Write("<td valign=
                        \"top\"><b>S.O.</b></td>");
     foreach(object obj in list)
     { StatManager sm = (StatManager)obj;
        context.Response.Write("<tr>");
        context.Response.Write("<td valign=\"top\">"
          + sm.DataVisita.ToString("d") + "</td>");
        context.Response.Write("<td valign=\"top\">"
          + sm.URLPagina.Replace("http://localhost
         /ioProgrammo/MyStatsTest/", "") + "</td>");
        context.Response.Write("<td valign=
                   \"top\">" + sm.IP + "</td>");
```

```
context.Response.Write("<td valign=
        \"top\">" + sm.Browser + "</td>");
context.Response.Write("<td valign=\"top\">"
    +sm.TempoDiEsecuzione.Milliseconds+"</td>");
context.Response.Write("<td valign=
        \"top\">" + sm.URLReferer.Replace(
    "http://localhost/MyStats/", "") + "</td>");
context.Response.Write("<td valign=\"top\">"
        + sm.SistemaOperativo + "</td>");
context.Response.Write("</tr>"); }
context.Response.Write("</table>"); }
```

il codice restituisce sempre true per il metodo *IsReusable* per indicare che l'istanza della classe può essere utilizzata da un'altra richiesta. Il punto cruciale della classe, però, sta nel metodo *ProcessRequest*, il reale responsabile della generazione dell'output. Come potete vedere, nel metodo si procede a recuperare le informazioni memorizzate nella cache che vengono infine scritte nel flusso restituito al client richiedente.

IMPOSTIAMO IL WEB.CONFIG

A questo punto, costruito il nostro modulo ed il nostro gestore http, dobbiamo in qualche modo collegarci all'applicazione Web per impostarne l'uso. Per far questo, apriamo il *web.config* dell'applicazione ed inseriamo le righe di seguito riportate:

```
<configuration>
<system.web>
  <httpModules>
    <add type="MyStats.StatModule,MyStats"
```

```
                              name="StatModule"/>
  </httpModules>
  <httpHandlers>
    <add verb="*" path="mystats.axd" type=
                   "MyStats.StatHandler,MyStats"/>
  </httpHandlers>
</system.web>
</configuration>
```

In questo modo dichiariamo l'utilizzo del gestore http e del modulo http attraverso l'indicazione del tipo, dell'assembly in cui esso è contenuto e del nome che gli vogliamo assegnare. In realtà è facoltativo specificare il nome, il cui scopo è l'opportuna indicazione per l'eventuale gestione di eventi nel *global.asax*.

CONCLUSIONI

In questo articolo abbiamo visto come, con poche e semplici istruzioni, sia possibile intervenire nella pipeline per catturare ed eventualmente modificare le informazioni continuamente scambiate tra il client, nel nostro caso il browser, ed il server web. Tutto questo era realizzabile anche prima dell'avvento di ASP.NET, attraverso l'uso delle estensioni *ISAPI* (o *NSAPI*), una buona conoscenza di C++ ed una buona dose di pazienza. Le possibili applicazioni di quanto illustrato sono molteplici, basti pensare che il framework stesso, come detto, ne fa ampio uso. Per tutti i chiarimenti su eventuali dubbi o problemi nello sviluppo dell'articolo, potete far riferimento al forum di ioProgrammo, o al mio blog riportato nel box laterale. Buon lavoro.

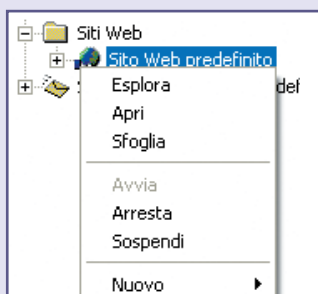
Fabio Cozzolino

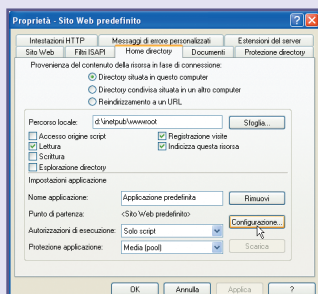
• **PROGRAMMARE MICROSOFT ASP.NET**
Dino Esposito
(Microsoft Press)
ISBN 88-8331-494-8.

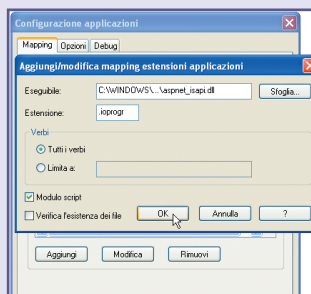
• **ASP.NET**
Stephen Walther
(APOGEO)
ISBN 88-7303-936-7
2002

IMPOSTARE UN'ESTENSIONE PERSONALIZZATA

Come creare un gestore per una risorsa personalizzata

> IIS

1 Aprire IIS (Internet Information Server), posizionarsi su Sito Web Predefinito ed accedere alle proprietà.

> LE PROPRIETÀ

2 Nelle proprietà accedere alla scheda "Home directory" e cliccare sul tasto "Configurazione".

> IL MAPPING

3 Nella scheda "Mapping" impostare "aspnet_isapi.dll" e l'estensione da mappare. Cliccare su conferma.

> IL WEB CONFIG

```
<configuration>
  <system.web>
    <httpHandlers>
      <add verb= "*" path=
     ".ioprog" type= "Handlers
        .ioProgrammoHandler,
         MyWebApplication"/>
    </httpHandlers>
  </system.web>
</configuration>
```

4 Nel web.config della nostra applicazione inseriamo la dichiarazione del nostro gestore http personalizzato.

PDF? Si grazie!!! facciamolo con Java

Utilizzeremo una libreria OpenSource: iText per creare file PDF in pochi passi. Vedremo come formattare il testo, inserire immagini, usare Adobe Actionscript e come firmare o proteggere i documenti

Manipolare file con una forma più complessa di un semplice file testo è sempre un'impresa ardua. I formati più diffusi hanno tipicamente licenze proprietarie perciò non riproducibili e in ogni caso spesso si tratta di formati di difficile comprensione. Quando si ha necessità di sviluppare applicazioni che producano in output un determinato tipo di file spesso e volentieri ci si scontra con difficoltà oggettive, il cui superamento si concretizza in complicate "acrobazie programmative". Il nostro scopo in questo articolo, sarà quello di individuare e gestire un formato di file universale che produca lo stesso output su qualunque piattaforma, non sia troppo limitato da licenze di vario genere, ci consenta di aggirare i limiti di cui sopra. Per le sue caratteristiche abbiamo scelto il *PDF (Portable Document Format),* il formato della Adobe che risponde alla maggior parte delle nostre esigenze. Ci rimane il problema di come Java possa produrre in output un file in formato PDF, ma in questo senso ci verrà in aiuto *iText,* una libreria opensource che per le sue caratteristiche si adatta perfettamente ai nostri scopi.

TI PRESENTO ITEXT

iText è una libreria sviluppata da Bruno Lowagie e Paulo Soares per la creazione di documenti da codice Java. È disponibile sotto le licenze *MPL (Mozilla Public License)* e *LGPL (Lesser General Public License),* e permette di creare documenti PDF, RTF e HTML. In questo articolo, come già detto ci occuperemo della creazione di file in formato PDF, anche se la maggior parte delle classi che andremo a creare, possono facilmente essere riutilizzate per la gestione degli altri formati supportati. Adobe relativamente alla specifica PDF 1.4 permette a terze parti di creare driver e software che come output hanno un PDF, perciò anche la questione delle licenze dovrebbe essere superata.

Conoscenze richieste
J2SE,

Software
J2SE SDK, iText 1.3

Impegno

Tempo di realizzazione

COME INIZIARE

iText è scaricabile dal sito http://www.lowagie.com/iText/, repository della versione curata da Bruno Lowagie. All'url http://itextpdf.sourceforge.net/ è invece possibile scaricare la versione curata da Paulo Soares. Ovviamente nel CD di ioProgrammo trovate tutte e due le versioni.

CIAO MONDO

Il nostro primo obiettivo sarà creare un programma Java che produca un file in formato PDF il cui contenuto sarà costituito dalle magiche parole: *"Hello World".* Il package principale che riguarda i PDF è *com.lowagie.txt.pdf,* all'interno del quale trovano posto diverse classi utili. Il punto di partenza sarà la classe *Document,* che descrive un documento generico, questo appare ovvio dal momento che le librerie *iText* non riguardano soltanto i PDF ma possono essere utilizzate anche per la produzione di altri tipi formati. Dopo aver istanziato un oggetto *Document,* istanzieremo la classe *PdfWriter* che provvederà, a fronte delle modifiche fatte sul documento, a serializzare l'oggetto sul filesystem. Ecco quindi il nostro *HelloWorld*

```
import com.lowagie.text.*;
import com.lowagie.text.pdf.*;
import java.io.*;
public class HelloWorldPDF {
public static void main(String[] args) {
System.out.println("Il primo PDF fatto con iText");
Document document = new Document();
try {
PdfWriter.getInstance(document,
          new FileOutputStream("HelloWorldPDF.pdf"));
document.open();
document.add(new Paragraph("Il primo PDF fatto con
                                         iText"));}
```

```
catch (Exception e) {
System.err.println(e.toString());}
document.close();}
}
```

Fig. 1: ***Il nostro PDF con iText***

Come potete vedere dal codice riportato, dopo aver istanziato *Document* e *PdfWriter* dobbiamo soltanto manipolare l'oggetto *Document* che rappresenta il vero e proprio documento PDF. In questo primo esempio abbiamo aggiunto un nuovo paragrafo, passando come parametro una stringa. Come sempre succede quando abbiamo a che fare con dei file dobbiamo chiudere il file (in questo caso il *Document*) per avere alla fine dell'esecuzione il flush del buffer creato dal nostro programma. Il formato standard della pagina creata è l'A4, *iText* supporta anche altre modalità, ad esempio ecco come creare un PDF in modalità *Landscape*

```
Document document = new Document(
                                PageSize.A4.rotate());
PdfWriter.getInstance(document, new
            FileOutputStream("ProvaLandscape.pdf"));
document.open();
document.add(new Paragraph("Un PDF in Landscape"));
document.close();
```

NON SOLO TESTO

Passiamo ora all'inserimento di alcune immagini all'interno del nostro PDF. Le immagini supportate da *iText* sono JPEG, PNG e GIF. Aggiungeremo due diversi formati di immagini al nostro PDF utilizzando la classe *Image*.

```
PdfWriter.getInstance(document, new
          FileOutputStream("Pdf_con_immagini.pdf"));
document.open();
Image gif = Image.getInstance("iop.gif");
gif.setAlignment(Image.MIDDLE);
Image jpeg = Image.getInstance("iop90.jpg");
jpeg.setAlignment(Image.LEFT);
Paragraph p=new Paragraph("Parola di Java!!!",new
                          Font(Font.TIMES_ROMAN, 28,
                        Font.ITALIC|Font.UNDERLINE));
p.setAlignment(Paragraph.ALIGN_RIGHT);
document.add(gif);
document.add(jpeg);
document.add(p);
```

In questo esempio abbiamo utilizzato diverse classi utili per formattare i documenti PDF. La prima classe importante su cui puntare l'occhio è *Image* che, come già detto, permette di gestire allo stesso modo immagini GIF, JPG e PNG. Usando il metodo statico *getInstance()* possiamo utilizzare allo stesso modo immagini di diversi formati. La seconda classe importante è *Paragraph*, l'abbiamo già incontrata nel primo esempio, ma questa volta l'abbiamo usata selezionando anche il font con cui il testo dovrà essere disegnato, per fare questo abbiamo usato un costruttore diverso. L'ultima nota riguarda l'allineamento di tutti e tre gli elementi grafici tramite il metodo *setAlignment()*, nel quale viene specificata la posizione.

Fig. 2: ***L'inserimento di diversi tipi di immagine all'interno del PDF***

PROTEGGERE I DATI

Un'interessante opportunità offerta dal formato PDF è quella di consentire di proteggere i documenti, questa possibilità si rivela particolarmente utile ad esempio quando vogliamo concedere solo a determinati utenti di potere leggere il contenuto dei file. La classe *PdfWriter* consente di settare una password per il documento, addirittura e offre addirittura la possibilità di diversificare tra lettore del PDF e autore.

```
PdfWriter writer = PdfWriter.getInstance(
     document,new FileOutputStream("Criptato.pdf"));
writer.setEncryption(PdfWriter.STRENGTH128BITS,
          "lettore", "scrittore", PdfWriter.AllowPrinting);
```

Dopo aver istanziato *PdfWriter* viene utilizzato il metodo *setEncryption()*, al quale vengono passati come argomento rispettivamente la robustezza della criptazione, la password per il lettore, la password per l'autore e i permessi che vengono concessi al lettore. Nel caso relativo all'esempio concedia-

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

mo soltanto il permesso di stampare il documento. Nel caso in cui il lettore volesse fare copia e incolla ciò non sarebbe permesso, perché disabilitato.

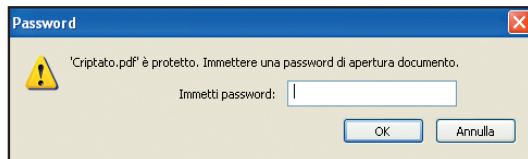



Fig. 3: **La richiesta che ci viene fatta dal nostro lettore quando incontra un PDF con password**

FIRMA DIGITALE DEI PDF

Si fa un gran parlare, recentemente, di firma digitale. Semplificando molto diciamo che grazie alla firma digitale è possibile apporre su un documento una sorta di sigla elettronica che è associata strettamente alla persona che firma. È praticamente l'equivalente della firma classica rapportata all'informatica. La firma digitale viene ad esempio utilizzato nelle email per apporre appunto una firma che conferma l'autore del messaggio. Questa tecnica può essere applicata anche al nostro caso. Infatti possiamo voler firmare i documenti che creiamo con una firma sicura. Prima di tutto creiamo una nostra chiave. Sul nostro computer, dentro la sottocartella *bin* della *HOME* di java troviamo un eseguibile, *keytool*, che tra le altre cose ci permette di creare una nostra chiave.

```
C:\Programmi\Java\jdk1.5.0_02\bin>keytool -genkey
    -keyalg RSA -alias ioprogrammo
-keypass articoloitext -keystore keystore.ks -dname
    "cn=Io Programmo, c=IT"
```

```
Seleziona Prompt dei comandi
C:\Programmi\Java\jdk1.5.0_02\bin>keytool -genkey -keyalg RSA -alias ioprogrammo
 -keypass articoloitext -keystore keystore.ks -dname "cn=Io Programmo, c=IT"
Immettere la password del keystore:  articoloitext

C:\Programmi\Java\jdk1.5.0_02\bin>dir
 Il volume nell'unità C non ha etichetta.
 Numero di serie del volume: 2C88-D539

 Directory di C:\Programmi\Java\jdk1.5.0_02\bin

04/05/2005  16.52    <DIR>          .
04/05/2005  16.52    <DIR>          ..
04/03/2005  03.12            49.274 appletviewer.exe
04/03/2005  03.12            49.263 apt.exe
04/03/2005  03.54            36.975 beanreg.dll
04/03/2005  03.12            49.272 extcheck.exe
04/03/2005  03.36            24.576 HtmlConverter.exe
04/03/2005  03.12            49.274 idlj.exe
04/03/2005  03.13            49.255 jar.exe
04/03/2005  03.13            49.276 jarsigner.exe
04/03/2005  03.13            49.248 java.exe
04/03/2005  03.13            49.269 javac.exe
04/03/2005  03.13            49.275 javadoc.exe
04/03/2005  03.13            49.269 javah.exe
04/03/2005  03.13            49.265 javap.exe
04/03/2005  03.13            49.250 javaw.exe
04/03/2005  03.36           127.078 javaws.exe
04/03/2005  03.13            49.274 jconsole.exe
04/03/2005  03.13            49.278 jdb.exe
04/03/2005  03.13            49.259 jps.exe
04/03/2005  03.13            49.265 jstat.exe
04/03/2005  03.13            49.268 jstatd.exe
04/05/2005  16.52             1.220 keystore.ks
04/03/2005  03.13            49.272 keytool.exe
04/03/2005  03.13            49.268 kinit.exe
04/03/2005  03.13            49.268 klist.exe
04/03/2005  03.13            49.266 ktab.exe
```

Fig. 4: **Generazione della chiave utilizzando keytool**

La nostra chiave ora è contenuta all'interno del file *keystore.ds*. Tramite il package standard di Java relativo alla sicurezza *(java.security)* caricheremo questa chiave e la inseriremo in un file PDF.

```
KeyStore ks = KeyStore.getInstance(
                    KeyStore.getDefaultType());
ks.load(new FileInputStream("keystore.ks"),
                    "articoloitext".toCharArray());
String alias = (String)ks.aliases().nextElement();
PrivateKey key = (PrivateKey)ks.getKey(alias,
                    "articoloitext".toCharArray());
java.security.cert.Certificate[] chain =
                    ks.getCertificateChain(alias);
PdfReader reader = new PdfReader("nonsegnato.pdf");
FileOutputStream fout = new
                    FileOutputStream("segnato.pdf");
PdfStamper stp = PdfStamper.createSignature(
                    reader, fout, '\0');
PdfSignatureAppearance sap =
                    stp.getSignatureAppearance();
sap.setCrypto(key, chain, null,
                    PdfSignatureAppearance.SELF_SIGNED);
sap.setReason("Autore del documento");
sap.setLocation("Roma");
sap.setVisibleSignature(new Rectangle(100, 100,
                    200, 200), 1, null);
stp.close();
```

Prima di tutto abbiamo è stato istanziato un *KeyStore*, nel quale è stata caricata la chiave che abbiamo creato precedentemente. Successivamente è stata creata una nostra chiave privata e di seguito un *Certificate*. Fatto questo, dobbiamo prendere un PDF, che non abbiamo precedentemente segnato, e produrre uno identico ma firmato. Dobbiamo allegare al PDF la nostra chiave con la classe *PdfSignatureApperance*, settando diverse informazioni. Così facendo nel PDF apparirà un'icona con le nostre informazioni, che testimonia il fatto che il PDF è stato creato da noi.

LINK IPERTESTUALI

All'interno di un documento PDF è possibile inserire elementi dinamici come ad esempio i link. Per inserire un link è necessario utilizzare la classe *Anchor*, che una volta inizializzata e inserita all'interno di un *Paragragh* permette di visualizzare un collegamento ipertestuale.

```
Paragraph paragraph = new Paragraph("Ecco alcuni
                    esempi di link");
    Anchor anchor1 = new Anchor("\nIL MIO SITO",
        FontFactory.getFont(FontFactory.HELVETICA, 12,
            Font.UNDERLINE, new Color(0, 0, 255)));
    anchor1.setReference("http://www.javastaff.com");
    anchor1.setName("URL");
    Anchor anchor2 = new Anchor("\nLA MIA EMAIL",
        FontFactory.getFont(FontFactory.HELVETICA, 12,
            Font.UNDERLINE, new Color(0, 0, 255)));
    anchor2.setReference("mailto:doc@javastaff.com");
    anchor2.setName("EMAIL");
```

IL FORMATO PDF

Il PDF è un formato che è stato creato intorno al 1991 dalla Adobe. Nei seguenti URL trovate informazioni riguanti la storia e i particolati tecnici di questo tipo di file.

http://www.prepressure.com/pdf/history/history01.htm
http://www.adobe.it/products/acrobat/adobepdf.html
http://www.planetpdf.com/developer/index.asp

```
Anchor anchor3 = new Anchor("\nFTP CON
                        APPUNTI DELLE DISPENSE",
    FontFactory.getFont(FontFactory.HELVETICA, 12,
            Font.UNDERLINE, new Color(0, 0, 255)));
anchor3.setReference("ftp://server");
anchor3.setName("FTP");
paragraph.add(anchor1);
paragraph.add(anchor2);
paragraph.add(anchor3);
document.add(paragraph);
```

Fig. 5 La firma digitale che viene inclusa nel documento PDF

All'interno del *Paragraph* abbiamo inserito tre diversi *Anchor.* Abbiamo specificato il testo che dovrà essere visualizzato, l'URL e un nome relativo. Se andiamo a vedere il PDF generato vedremo che verranno visualizzati dei veri e propri link, come quelli che siamo abituati a vedere nelle pagine web. Inoltre è prevista la possibilità di richiamare queste informazioni all'interno del documento diverse volte. Praticamente potremmo inserire una bibliografia o diversi indirizzi email, che magari all'interno di un testo vengono utilizzati più volte e riferirci a questi tramite il nome relativo che abbiamo deciso

```
Anchor anchor2 = new Anchor("L'email dell'autore",
    FontFactory.getFont(FontFactory.HELVETICA, 12,
            Font.NORMAL, new Color(0, 0, 255)));
anchor2.setReference("#EMAIL");
```

ACROBAT JAVASCRIPT

Acrobat Javascript è un vero e proprio linguaggio di scripting, aderente alle specifiche del Javascript che permette di ottenere la stessa dinamicità delle pagine web all'interno di documenti PDF. *iText* supporto abbondantemente l'Acrobat JavaScript. Il nostro primo esempio riguarderà l'apertura di una classica finestra popup

```
Paragraph p = new Paragraph(new Chunk("Hello
                                        Window")
.setAction(PdfAction.javaScript("app.alert('Hello
                            Window');\r", writer)));
document.add(p);
```

In questo caso per aggiungere il codice Javascript alla pagina PDF abbiamo usato il metodo *setAction()* della classe *Chunk*. Così facendo abbiamo impostato un'azione per un oggetto grafico, il codice Javascript verrà eseguito in reazione all'evento *OnClick*. Allo stesso modo in cui abbiamo aggiunto un popup possiamo aggiungere dei link per avviare delle vere e proprie applicazioni esterne presenti sul PC.

```
Paragraph p = new Paragraph();
   Chunk notepad=new Chunk("Notepad\n")
           .setAction(new PdfAction("C://WINDOWS
                 /notepad.exe", null, null, null));
   Chunk er=new Chunk("Esplora risorse\n")
           .setAction(new PdfAction("C://WINDOWS
                 /explorer.exe", null, null, null));
   Chunk calc=new Chunk("Calcolatrice\n")
           .setAction(new PdfAction("C://WINDOWS
             /system32/calc.exe", null, null, null));
   p.add(notepad);
   p.add(er);
   p.add(calc);
```

Costruendo una *PdfAction* e passando come argomento l'eseguibile di un applicazione è possibile avviare un programma esterno. Il lettore di PDF prima di proseguire chiederà l'utente vuole eseguire un programma esterno, questo ovviamente per motivi di sicurezza.

CONCLUSIONI

Il livello di questa applicazione non è chiaramente professionale ma è sufficiente per mostrarvi con quanta semplicità sia possibile manipolare file PDF in Java grazie ad iText. Inoltre dobbiamo dire che la libreria, soprattutto dal punto di vista grafico espone alcune altre caratteristiche interessanti che potrebbero risultare utili in applicazioni più complesse. Questa libreria può essere tranquillamente usata in applicazione server come JSP/Servlet e quindi possiamo creare dinamicamente dei PDF da restituire all'utente. Inoltre è disponibile anche una versione .NET, *iTextdotNET*, per poter utilizzare questa libreria con linguaggi diversi da Java. Insomma *iText* è un'ottima libreria da includere nei nostri programmi per poter creare applicazioni portabili (Java) con documenti portabili (PDF).

Federico Paparoni

Fig. 6: Come vengono visualizzati i diversi link che possiamo inserire nel nostro documento

NOTA

COSA C'È OLTRE ITEXT?

iText non è l'unica libreria Java che permette di manipolare i PDF. Esistono infatti altre due valide alternative. La prima è PDFBOX, libreria che permette di accedere a tutti i componenti di un PDF, sia in scrittura che in lettura. La seconda è gnujpdf, package Java rilasciato sotto licenza LGPL che permette di creare e stampare PDF.
http://www.pdfbox.org/
http://gnujpdf.sourceforge.net/

Hacker: addio al Reverse Enginering

Gelosi del vostro codice? Ecco a voi una panoramica delle più conosciute tecniche di offuscamento del bytecode Java, per scoraggiare eventuali tentativi di Reverse Engineering con l'aiuto di RetroGuard

Tra i punti di forza che hanno fatto la fortuna della tecnologia Java c'è sicuramente l'indipendenza, quindi la portabilità, del compilato dalla piattaforma. A differenza di tutti gli altri linguaggi di programmazione, come ad esempio il C, dove la compilazione genera il codice sorgente contenente il linguaggio macchina del processore su cui il programma dovrà essere eseguito, la codifica Java crea dei particolari file binari contraddistinti dall'estensione *".class"*.
La Java Virtual Machine non ha assolutamente alcuna conoscenza del linguaggio di programmazione, essa si limita unicamente ad interpretare i *class files*.
Le componenti più importanti contenute all'interno di questi file binari sono il *bytecode*, ovvero l'insieme delle istruzioni da dare in pasto alla JVM durante la fase di esecuzione di un'applicazione ed una tabella dei simboli. Un programmatore ha sempre il modo di decompilare qualsiasi tipo di programma. Solitamente i processi di decompilazione sono abbastanza lunghi e richiedono abilità ed esperienza. Non si può dire la stessa cosa quando si prendono in considerazione le classi Java che sono molto più semplici da decodificare, poiché presentano un "macro-linguaggio" semplice e la maggior parte delle istruzioni sono eseguite dalle librerie standard. Pertanto, esistono molte applicazioni facilmente reperibili dalla rete e solitamente free, che permettono la decompilazione delle classi anche a programmatori che non hanno alcuna conoscenza del funzionamento interno della JVM.
Questi attacchi, il cui scopo è la violazione della proprietà intellettuale del software, potrebbero rivelarsi costosi, in quanto permetterebbero, ad eventuali competitori, l'annullamento in tempi ridotti dei vantaggi tecnologici sviluppati. Per scoraggiare la decodifica delle classi vengono solitamente usati dei tool, di facile utilizzo, conosciuti come *"offuscatori del bytecode"*, che hanno l'obiettivo di complicare la lettura del codice decompilato attraverso l'applicazione di particolari algoritmi.
Prima di approfondire la conoscenza di questi applicativi sarà meglio andare ad analizzare il loro funzionamento interno, per capire il lavoro da essi svolto.

COME INIZIARE

1) È indispensabile avere installato sul computer Java 2 Standard Edition SDK 1.1 o superiore.
É preferibile utilizzare l'ultima versione disponibile dal sito http://java.sun.com.

2) Scaricare il tool di offuscamento bytecode RetroGurd v2.0.2 dal sito http://www.retrologic.com/retroguard-download.html.
É possibile installare il tool, indipendentemente dalla piattaforma sulla quale sarà utilizzato, estraendo il contenuto dell'archivio scaricato sul file system (generalmente in *<SYSTEM_DRIVE_ROOT>\Programmi* per OS Windows oppure */opt* per OS Unix/Linux).
La cartella retroguard-v2.0.2 sarà creata automaticamente. Aggiungere alla variabile d'ambiente *CLASSPATH* il percorso assoluto del file *retroguard.jar.*
La documentazione di RetroGuard è consultabile online all'indirizzo http://www.retrologic.com/retroguard-docs.html.
Il processo di offuscamento può essere applicato soltanto ad archivi con estensione *".jar"*.
Per crearli è possibile usare l'eseguibile *jar* contenuto nella directory *bin* della Java Home.
Di seguito è mostrato un esempio che descrive la notazione standard per la creazione di un archivio contenente tre classi:

```
cvf nomeArchivio.jar
        Classe1.class Classe2.class
                      Classe3.class
```

L'esecuzione del comando senza alcun argomento ne descrive l'uso in maniera dettagliata.

TECNICHE DI OFFUSCAMENTO

Rilevando che il processo di offuscamento non garantisce una completa protezione del codice, a fronte di attacchi di reingegnerizzazione, possiamo definire tale tecnica una trasformazione di un programma in un altro programma funzionalmente equivalente, ma di difficile comprensione ed analisi dal punto di vista umano.
Si parla di offuscamento "funzionalmente equivalente" quando l'applicazione originale e quella offuscata producono i medesimi valori in uscita, a parità di condizioni iniziali.
Ovviamente bisogna anche tener d'occhio le performance, sia in termini di velocità in esecuzione che di spazio disco utilizzato a runtime.
Le più conosciute metodologie usate per rendere il codice meno intelligibile sono: *layout obfuscation, control obfuscation, data obfuscation* e *preventive transformation*.
Per meglio comprendere le trasformazioni apportate da ciascun algoritmo prendiamo in considerazione il seguente metodo statico che restituisce l'area di un triangolo:

```
public static double areaTriangolo(int base, int altezza)
{
    double area = (double)(base * altezza) / 2;
    return area;
}
```

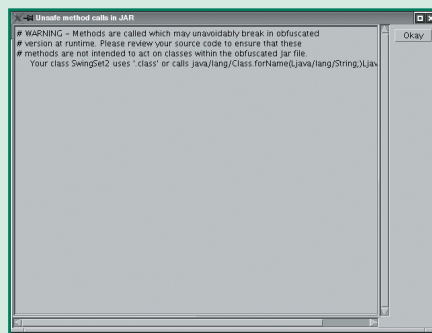
Layout obfuscation (o di superficie): consiste nella modifica dell'aspetto "fisico" del codice, eliminando le parti non necessarie alla sua esecuzio-

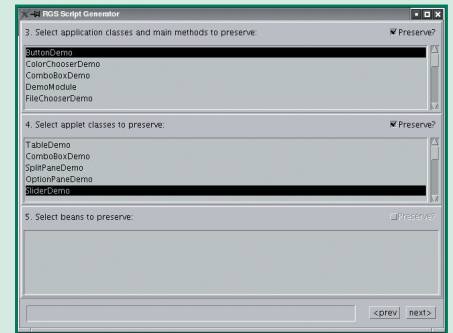
CODICE OFFUSCATO IN SEI PASSI

Vediamo come Reroguard ci viene in aiuto nella creazione del file .RGS necessario ad individuare i punti critici che non possono essere offuscati

› CREIAMO UNO SCRIPT.RGS

1 Inseriamo il path assoluto del file SwingSet2.jar, contenuto nelle demo della JDK.

› REFLECTION WARNINGS

2 Il tool ci consiglia un'analisi delle invocazioni via reflection, per evitare errori a runtime.

› PRESERVARE MAIN, APPLET E BEAN

3 Le classi contenenti main, applet e bean sono riconosciute ed escluse dal processo.

› PRESERVARE CLASSI AGGIUNTIVE

4 Possiamo specificare, in modo dettagliato, altre classi da escludere dal processo.

› PRESERVARE METODI E ATTRIBUTI

5 Specifichiamo particolari metodi e variabili su cui non applicare l'offuscamento.

› IMPOSTAZIONI GENERALI

6 Impostazioni generali da applicare a tutte le classi e generazione script .rgs. Abbiamo terminato.

ne, come ad esempio le informazioni di debugging. Inoltre questo algoritmo rimuove tutti i tipi di formattazione e sostituisce i nomi originali di classi, metodi e variabili con altri casuali o addirittura con un numero differente di caratteri *"underscore"*.
Il nostro metodo *areaTriangolo* diventerà:

```
public static double ____(int ___, int _____)
{
   double ______ = (double)(___ * _____) / 2;
   return ______;
}
```

Control obfuscation: ha come obiettivo la manipolazione del flusso di esecuzione del programma, inserendo, ad esempio, espressioni o loop. In base a ciò, il metodo *areaTriangolo* potrà essere modificato nel seguente modo:

```
public static double areaTriangolo(int base, int altezza)
{
    double area = 0d;
    for (int i = 0; i < base; i++)
    {
        area += altezza;
    }
    area = area / 2;
    return area ;
}
```

Data obfuscation: serve a rallentare l'azione di decompilazione attraverso la sostituzione di operazioni semplici con altre più articolate e complesse. Ecco l'offuscamento di tipo *Data* applicato al metodo *areaTriangolo*, precedentemente trasformato con *Control obfuscation:*

```
public static double areaTriangolo(int base, int altezza)
{
    double area = 0d;
    base *= 5 << 1;
    for (int i = 0; i < base; i += 20 >> 1)
    {
        area += altezza * 2 >> 1;
    }
    area = area / 2;
    return area ;
}
```

Di seguito è mostrato il codice dopo l'applicazione dei tre algoritmi finora descritti:

```
public static double ____(int ___, int _____)
{
   double ______ = 0d;
   ___ *= 5 << 1;
   for (int _ = 10; _ < ___; _ += 20 >> 1)
   {
      ______ += _____ * 2 >> 1;
   }
   ______ = ______ / 2;
   return ______;
}
```

Preventive transformation: a differenza delle tre tecniche sopra descritte, quella di tipo *Preventive* non ha l'obiettivo di rendere il codice meno leggibile, ma tenta di sfruttare i punti deboli dei tool di decompilazione. Un tipico esempio è l'introduzione nel bytecode di istruzioni che non saranno mai raggiunte dal flusso di esecuzione che, però, hanno la capacità di fare "ingarbugliare" alcuni decompilatori.
L'aggiunta di un'istruzione dopo la chiamata di un return è la più classica di queste trasformazioni. Sebbene molti tool di offuscamento, come ad esempio *Crema*, applichino questo tipo di protezione, il libro *"The Java™ Virtual Machine Specification"* ne proibisce l'uso, dichiarato pericoloso per le future implementazioni della JVM.

RETROGUARD 2.0.2

RetroGuard è uno dei più conosciuti strumenti di offuscamento del bytecode Java. Il suo sviluppo, iniziato nell'anno 1998, grazie ad un progetto della Retrologic Systems, è arrivato, al momento della stesura di questo articolo, alla versione 2.0.2. Esistono due differenti distribuzioni: una con licenza GPL, per uso non commerciale, e l'altra con licenza annuale al costo di $139 (U.S. Dollar), per uso commerciale. Le funziona-

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

DENTRO LA JVM

Così come ogni comune unità di elaborazione, la JVM definisce delle aree dati a runtime che sono utilizzate durante l'esecuzione dei programmi. Ad esclusione dei metodi nativi, l'indirizzo dell'istruzione in corso di esecuzione è memorizzato in un registro, chiamato contatore di programma (o program counter, PC). Ogni thread che viene creato inizializza automaticamente una struttura dati dinamica privata detta stack (pila), una coda del tipo last-in first-out. Per indicare il trasferimento di dati da e verso lo stack, sono usati dei particolari termini gergali: per push si intende la memorizzazione di un dato nello stack e per pop la sua estrazione. Quando, invece, istanziamo un oggetto, questo viene memorizzato in un'altra area detta heap. Differentemente dallo stack, la heap zone è un'area comune a tutti i thread in esecuzione ed è, inoltre, il campo d'azione del garbage collector, il cui compito è la deallocazione degli oggetti non più referenziati. Quelli appena introdotti sono solo alcuni accenni alle strutture fondamentali della JVM, ma è utile sapere che ne esistono anche altre. Per chiunque volesse approfondire il funzionamento interno della JVM è possibile scaricare dal sito http://java.sun.com il libro "The Java™ Virtual Machine Specification" di Tim Lindholm e Frank Yellin.

lità sono identiche per entrambe le distribuzioni. É interamente scritto in Java (Java 5 compliant) e genera esclusivamente bytecode pienamente compatibile con le specifiche della JVM, pertanto non supporta trasformazioni di tipo *Preventive*. Tra le nuove features introdotte in quest'ultima release, va sottolineata la possibilità di essere facilmente integrato in regolari processi di *quality-assurance* e *build* tramite l'uso di *ant* (*http://ant .apache.org*). Prima di iniziare il processo di offuscamento è necessario creare uno script file, solitamente identificato dall'estensione *".rgs" (RetroGuard script)*. Generalmente non è possibile applicare le tecniche di offuscamento su un intero progetto, in quanto alcuni elementi devono essere accessibili dall'esterno. Ad esempio, non è possibile applicare la *layout obfuscation* né al nome del metodo *main,* né tanto meno alla classe che lo contiene: entrambi dovranno essere obbligatoriamente preservati per garantire il corretto funzionamento dell'applicazione. Il contenuto di un file *rgs* è sostanzialmente la lista degli elementi da non offuscare e deve essere creato tramite il formalismo di *Backus-Naur (BNF)*. Individuare tutti gli elementi da preservare dall'offuscamento non è sempre semplice, soprattutto quando ci si trova a lavorare su progetti di grandi dimensioni. Il tool ci viene in aiuto mettendoci a disposizione un'intuitiva interfaccia grafica, che analizza l'archivio da offuscare, al fine di individuare in maniera automatica i potenziali punti "critici" su cui non applicare alcun tipo di trasformazione.
Questa GUI, che prende il nome di *RetroGuard Script Generator,* ha la funzione di guidarci step by step alla creazione dello script da utilizzare in fase di offuscamento, senza presupporre alcuna conoscenza della grammatica BNF. Una volta ottenuto lo script *rgs*, non ci rimane che avviare il processo di offuscamento eseguendo il comando:

```
java RetroGuard [INPUT-JAR [OUTPUT-JAR [SCRIPT
                                        [LOGFILE]]]]
```

dove:

- **INPUT-JAR** è il nome dell'archivio contenente le classi originali da offuscare (default: *in.jar*).
- **OUTPUT-JAR** è il nome dell'archivio contenente le classi offuscate, che sarà creato da *RetroGuard* (default: *out.jar*).
- **SCRIPT** è lo script creato precedentemente attraverso *RetroGuard Script Generator* (default: *script.rgs*).
- **LOGFILE** è il file di log, creato da *RetroGuard*, contenente le specifiche di offuscamento (default: *retroguard.log*). Solo ed esclusivamente attraverso questo file si può risalire dall'archivio offuscato a quello originale.

CONCLUSIONI

In questo articolo abbiamo visto come un programmatore malizioso possa facilmente arrivare a decompilare il bytecode Java. Come ribadito in precedenza, le tecniche di offuscamento non rappresentano una protezione assoluta delle tecnologie e metodologie contenute all'interno del codice compilato. L'applicazione di tali trasformazioni rappresentano un ostacolo, solitamente costoso in termini di tempo, che ha la finalità di rendere più resistente il bytecode da attacchi di *Reverse Engineering*.
In base all'efficacia degli algoritmi utilizzati è possibile valutare la potenza dell'offuscamento e distinguere tre tipologie di trasformazioni: ad alta, media e bassa potenza.
Se, ad esempio, durante lo sviluppo si fa un uso massiccio di invocazioni tramite *Reflection,* l'indice di potenza dell'offuscamento potrebbe risultare basso, a causa di un uso ridotto di trasformazioni di tipo Layout. Quando, come in questi casi, l'offuscamento risulta essere inadeguato alle finalità prefisse, si ha la tendenza di utilizzare metodi alternativi più efficienti come il *"cripting"* delle classi.
Nei prossimi numeri analizzeremo i vari metodi di cripting, confrontandoli con quelli di offuscamento in modo da distinguere quello che meglio si addice alle nostre esigenze.

Fabrizio Fortino

Specifiche ufficiali della Java Virtual Machine:
http://java.sun.com/docs/books/vmspec/

Tools di decompilazione:
http://www.brouhaha.com/~eric/software/mocha/
http://kpdus.tripod.com/jad.html
http://members.fortunecity.com/neshkov/dj.html
http://www.bysoft.se/sureshot/cavaj/

Tools di offuscamento:
http://www.retrologic.com/
http://www.yworks.com/en/products_yguard_about.htm
http://proguard.sourceforge.net/
http://sourceforge.net/projects/javaguard/

BACKUS-NAUR FORM

La BNF è un metalinguaggio utilizzato per descrivere la sintassi di un generico linguaggio. É stato utilizzato il termine "generico" in quanto è possibile applicare tale notazione ad ogni tipo di linguaggio e non solo a quelli strettamente legati a contesti informatici o tecnologici. La BNF fu inventata originariamente da John Backus per descrivere il linguaggio Algol60 e fu revisionata nel corso degli anni da Peter Naur. Pertanto l'acronimo che in un primo momento era tradotto in *"Backus normal form"* fu modificato, su proposta di Donald Knuth, in *"Backus-Naur Form"*.
Per descrivere un dato linguaggio vengono utilizzati i seguenti metasimboli: "::=" (definito come), "|" (oppure), "<>" (contengono il nome di una categoria). La notazione BNF di base è descritta dal seguente formalismo: <nome> ::= <regole>, dove nome è detto simbolo non terminale mentre le regole che non appaiono mai nella parte sinistra sono dette simboli terminali.
Un tipico esempio è la definizione della regola per descrivere la dichiarazione di package di una classe del linguaggio Java:

```
<package_statement>  ::=
          "package" package_name  ";"
```

Metti il turbo a Java

Introduzione a Javolution. Limitiamo la memoria utilizzata dal nostro programma, implementando strutture di memoria più veloci, performanti e facilmente serializzabili

Le prestazioni di Java e del suo Garbage Collector sono da sempre fonte di aspre discussioni in forum e newsgroups fra i fan di questo linguaggio. I loro avversari naturali, gli sviluppatori C++. Chi ha dimestichezza con i meccanismi di gestione della memoria sa che il Garbage Collector è un processo a bassa priorità che si occupa di gestire la frammentazione della memoria. Si tratta di un processo molto costoso in termini di occupazione del processore perché si occupa appunto di spostare blocchi di memoria da una posizione ad un'altra, "interrompendo" il flusso principale del programma per effettuare delle operazioni che non sono direttamente collegate alla sua esecuzione ma sono necessarie per gestire correttamente la memoria.

spazi affinché la memoria sia scarsamente frammentata. L'operazione di deallocare un oggetto e rendere disponibile la memoria da esso lasciata libera per altre operazioni va sotto il nome di "riciclo" dell'oggetto. Inoltre all'*heap* esiste un secondo tipo di segmento di memoria: lo *stack*. In uno *stack* vengono tipicamente allocati le funzioni e i propri parametri. Lo *stack* è tipicamente gestita in maniera LIFO. Ovvero l'ultima funzione allocata è anche la prima ad uscire. Ad esempio viene allocato il *main*, poi la funzione *func1*, quando la funzione *func1* ha terminato il suo compito lo spazio ad essa riservato viene deallocato e lo *stack* ritorna alle condizioni originali. Questo significa che nello *stack* non c'è frammentazione della memoria.

COME FUNZIONA IL GARBAGE COLLECTOR

Tipicamente un oggetto viene creato in seguito alla chiamata dell'operatore *new*. Creare un oggetto significa allocare la memoria necessaria a contenerlo. Possiamo immaginare la memoria come un nastro verticale diviso in tre segmenti: *Heap, Stack, Text*. Normalmente si sceglie di allocare agli oggetti uno spazio di memoria all'interno dell'*Heap*. L'allocazione avviene in maniera "non sequenziale", ovvero lo spazio viene riservato prelevandolo da un'area disponibile accedendo ad essa secondo un particolare algoritmo di gestione. Quando l'oggetto non è più referenziato da nessuna parte nel programma, la memoria ad esso riservata viene distrutta e resa disponibile ad altri oggetti. Questa operazione di allocazione e deallocazione della memoria come potete immaginare crea dei buchi nella memoria stessa a causa della diversa dimensione degli spazi allocati. A questo punto entra in gioco il Garbage Collector che si occupa di ricollocare gli

LA RIVOLUZIONE

"Javolution" si propone subito come ambizioso progetto per una rivoluzione nel mondo Java. Le premesse ci sono tutte: ampia compatibilità, da J2ME a J2EE 1.5 e un miglioramento delle prestazioni notevole. Questo avviene implementando un diverso meccanismo di gestione della memoria che consente di "riciclare" gli oggetti in modo più efficiente. Talmente efficiente che i test affermano che si possono ottenere prestazioni superiori al normale di oltre il 70%. Inoltre mette a disposizione delle strutture iperveloci che consentono di abbassare i tempi di esecuzione di un programma proprio grazie a un diverso uso dell'allocazione/deallocazione degli oggetti in memoria. Oltre alla classica gestione dell'*Heap*, Javolution ci mette a disposizione altri metodi di "riciclo degli oggetti":

- **Stack:** gli oggetti vengono riciclati quando non sono più referenziati nel *PoolContext* che li utilizza

- **Hold:** gli oggetti non vengono riciclati e tornano al loro stato originale quando un counter interno raggiunge lo 0.

In particolare il primo metodo è quello più interessante. Utilizzando lo *stack* gli oggetti che appartengono allo stesso *PoolContext* vengono allocati nel segmento *Stack* della memoria e vengono distrutti con la stessa logica Lifo quando il *PoolContext* in cui sono utilizzati termina con una chiamata *"exit"*. Questo evita da un lato la frammentazione della memoria, dall'altro che intervenga il Garbage Collector risparmiando così preziosi cicli di CPU. Per contro ci sono alcune regole da seguire affinché tutto funzioni: prima di tutto dovrà essere creato un "contesto di oggetti", ovvero un'insieme di oggetti etichettato con un nome. Gli oggetti appartenenti a un certo contesto non possono essere referenziati da oggetti appartenenti a un contesto diverso. Questo appare ovvio, se infatti ci fosse un riferimento a un certo oggetto al di fuori del contesto, questo non potrebbe essere distrutto con tanti saluti al LIFO. Viceversa, quando un thread che usa un certo contesto di oggetti termina, tutto viene distrutto in maniera automatica e non c'è frammentazione della memoria.
Creare un *PoolContext* è semplicissimo

```
PoolContext.enter();
try{
        de = Updater.updateAll("c:\");
}
finally
{ PoolContext.exit();
}
```

Fatto questo tutti gli oggetti creati da un *ObjectFactory* saranno gestiti nello *stack*. Va da sé che ogni classe in cui pensiamo di utilizzare oggetti di questo tipo deve implemetare. All'interno di un *PoolContext* tutti gli oggetti creati attraverso un *ObjectFactory* seguono una logica di tipo *stack*. Tutte le classi che intendono usare oggetti in questo modo devono implementare un *ObjectFactory* come segue:

```
static final ObjectFactory FACTORY = new
                                    ObjectFactory() {
    public Object create() {
        return new FileElement();
    }
};
```

Per costruire un nuovo oggetto basta chiederne un'istanza:

```
FileElement fe = (FileElement)
                            FileElement.FACTORY.object();
```

LISTE VELOCI

Oltre al supporto al realtime, Javolution ci offre una serie di strutture dati molto interessanti. Nel package *javolution.util* troviamo, infatti, versioni *"fast"* di molte classi già esistenti in Java. più veloci sia per quanto riguarda le prestazioni, sia per la maggiore facilità di utilizzo.

- **FastCollection**, una collezione di oggetti, da consultare attraverso un *fastIterator*, ovviamente.
- **FastList**, lista di oggetti, dinamica e particolarmente indicata in associazione con il riciclo di oggetti attraverso *PoolContext*.
- **FastMap**, simile a *java.util.LinkedHashMap* ma con prestazioni elevate.

In più, troviamo due interessanti implementazioni di funzioni spesso non disponibili in piattaforme minori:

- **MathLib,** assicura la portabilità di funzioni matematiche su tutte le piattaforme che supportano *javolution*. Interessantissima per esempio in J2ME CLDC 1.0, che ha non pochi problemi con la matematica!
- **Reflection,** supporto alla *reflection,* molto utile su piattaforme che da sole non la permettono.

Interessanti? Sicuramente sì; ad esempio per rappresentare la struttura di una directory, possiamo utilizzare una *FastList:*

```
public class DirectoryElement extends FileElement {
  private FastList list;
  static final ObjectFactory FACTORY = new
                                    ObjectFactory() {
     public Object create() {
       return new DirectoryElement();
     }
  };
  public DirectoryElement() {
      super();
      this.list=FastList.newInstance();
      this.isDirectory=true;
  }
  public void add(FileElement e){
     list.add(e);
  }
```

Per istanziare un nuovo oggetto *DirectoryElement* in maniera pulita basterà passare per l'*ObjectFactory,* come abbiamo già visto:

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

```
DirectoryElement rootD = (DirectoryElement)
                    DirectoryElement.FACTORY.object();
```

La stessa cosa avviene, ma ancora più facilmente, quando chiediamo una nuova istanza di *FastList:*

```
        this.list=FastList.newInstance();
```

Utilizzare una *FastList* è semplicissimo, come vediamo da uno stralcio dell'interfaccia grafica, *DesktopSearcherView.java,* nel punto in cui si visualizzano i risultati della ricerca, che *Searcher* ci restituisce in una *FastList:*

```
FastList Fl = Searcher.Search(
                        this.jTextField1.getText());
ListIterator iter = Fl.listIterator();
        int i=0;
        while (iter.hasNext()){
            array[i++][0]=(String)iter.next();
        }
```

Il meccanismo quindi è lo stesso di una normale *List,* che possiamo scorrere attraverso un *ListIterator*.

Le prestazioni sono effettivamente più elevate. Potete verificarlo personalmente, eseguendo il benchmark:

java -jar javolution.jar perf

LA PERSISTENZA

Una delle features più interessanti di Javolution è la serializzazione in XML, non tanto per il formato utilizzato per il salvataggio, ma soprattutto per via delle prestazioni promesse. Il processo di serializzazione-deserializzazione si basa infatti su *RealtimeParser,* XML parser con prestazioni 3-5 volte più veloci dei concorrenti, che sicuramente non dimenticheremo di prendere in considerazione anche in futuro. Ma vediamo come sia possibile serializzare i nostri oggetti:

```
new ObjectWriter().write(this, new File OutputStream(
     "/home/ronzola/cache/"+ Calendar.getInstance()
                        .getTimeInMillis()+".xml"));
```

Con questa unica linea, magari circondata da un blocco *try-catch,* andiamo a salvare su file l'intero oggetto *DirectoryElement.* Oltretutto salviamo l'oggetto in una maniera che potrebbe risultarci parecchio utile se un giorno volessimo implementare un processo di aggiornamento dei dati indicizzati, dato che il nome del file è nel formato *timestamp.xml.* Così basterà caricare i file in ordine alfabetico e avremo le cartelle nell'ordine cronologico in cui sono state analizzate. Dobbiamo infatti scegliere in che maniera far salvare l'oggetto. In cambio di questo piccolo sforzo avremo una completa gestione dell'xml prodotto. Nell'oggetto da serializzare dobbiamo definire un *XmlFormat:*

```
protected static final XmlFormat DIRECTORY_XML =
            new XmlFormat(DirectoryElement.class){
   public void format(Object obj, XmlElement xml) {
      DirectoryElement g = (DirectoryElement) obj;
      FILE_XML.format(g, xml);
      xml.getContent().addLast(g.list);
   }
   public Object parse(XmlElement xml) {
      DirectoryElement g = (DirectoryElement)
                              FILE_XML.parse(xml);
      g.list = (FastList)xml.getContent().removeFirst();
      return g; }
};
```

Abbiamo dovuto descrivere sia il metodo per scrivere il file, *format,* sia quello per parsarlo in fase di deserializzazione, *parse(XmlElement xml).* In questo caso ci preoccupiamo di serializzare solo la *FastList* di file contenuti dalla directory aggiungendola attraverso il metodo *addLast.* ma le possibiliàà sono ben più ampi. Abbiamo il controllo sul namespace e di impostare valori per attributi, come vediamo da queso esempio tratto dalle api *javolution:*

```
ArrayList names = new ArrayList();
names.add("John Doe");
names.add("Oscar Thon");
names.add("Jean Bon");
ObjectWriter ow = new ObjectWriter();
ow.setNamespace("", "java.lang"); // Default
              namespace for java.lang.* classes
ow.write(names, new FileOutputStream(
                            "C:/names.xml"));
```

Ed ecco un esempio di file XML prodotto:

```
 <root:java.util.ArrayList xmlns:root="java:"
                           xmlns="java:java.lang">
    <String value="John Doe"/>
    <String value="Oscar Thon"/>
    <String value="Jean Bon"/>
 </root:java.util.ArrayList>
```

IL PROGRAMMA

Abbiamo visto come far girare i nostri thread in un contesto che ne migliori le prestazioni, come utilizzare le veloci *FastList* e come serializzare i nostri oggetti su file XML. Applichiamo questi concetti a un nostro progetto. Le funzionalità che vogliamo offrire sono due:

- indicizzare le immagini del nostro disco

- poter eseguire ricerche mirate nel repository creato con l'indicizzazione.

La prima funzionalità la implementiamo in *Updater.java,* che potete trovare nel cd allegato alla rivista. In pratica esploriamo l'albero delle directory ricorsivamente, creandoci delle *DirectoryElement* contenenti una lista di *FileElement:*

```
public static DirectoryElement updateAll(String root){
        File descriptor = new File(root);
        DirectoryElement rootD = (DirectoryElement)
                        DirectoryElement.FACTORY.object();
        rootD.name = descriptor.getName();
        rootD.path = descriptor.getParent();
        rootD.timestamp = descriptor.lastModified();
        if (root.indexOf("DesktopSearcherCache")>-1)
                                              return rootD;
        File[] dir = descriptor.listFiles(new SuffixFilter());
        try{
        for (int i=0;i<dir.length;i++){
                if (dir[i].isDirectory()) {
                   DirectoryElement da = Updater.updateAll(
                                   dir[i].getAbsolutePath());
                        da.write();
                        da=null; }
                else {
                        FileElement fe = (FileElement)
                                   FileElement.FACTORY.object();
                        fe.name = dir[i].getName();
                        fe.path = dir[i].getParent();
                        fe.timestamp = dir[i].lastModified();
                        rootD.add(fe); } }
    } catch(Exception e){
    }
    return rootD;
}
```

Tutte le cartelle analizzate vengono serializzate nel formato *timestamp.xml,* salvandole nella cartella *DesktopSearcherCache*, che ci preoccupiamo di non andare a indicizzare. Probabilmente esistono moltissime soluzioni più valide e prestanti, ma sicuramente con questo metodo siamo riusciti a stressare a dovere il PC e testare la stabilità di queste librerie. Nei nostri test, in cui abbiamo indicizzato solo una parte del disco, siamo riusciti ad analizzare 3,6 Giga in meno di 20 secondi, andando a scrivere ben 9651 file XML. I tempi di ricerca nelle directory serializzate sono ben più veloci: in 4-5 secondi abbiamo tutte le immagini che contengono nel nome una particolare stringa pronte per essere visualizzate. Vediamo come avviene la ricerca:

```
public static FastList Search(String s){
    DirectoryElement a;
    FastList fileList=FastList.newInstance();
    File descriptor = new File(
                              "/DesktopSearcherCache/");
    File[] dir = descriptor.listFiles();
    try{
         for (int i=0;i<dir.length;i++){
            if (!dir[i].isDirectory()) {
                  a = (DirectoryElement) new ObjectReader()
                            .read(new FileInputStream(dir[i]));
                  fileList.addAll(a.searchByName(s));
                  a = null;
            }
         }
      } catch(Exception e){
                System.out.println("Errore in Apertura: "+e);
      }
      return fileList;
}
```

Praticamente andiamo a caricare tutte le cartelle deserializzandole, e su ognuna eseguiamo il metodo *searchByName(s)* che vediamo qui sotto:

```
public FastList searchByName(String s){
    FastList fileList=FastList.newInstance();
    ListIterator iter = list.listIterator();
    while (iter.hasNext()){
       Object elemento = iter.next();
       if (!((FileElement)elemento).isDirectory){
          String el = ((FileElement)elemento).name;
          if (el.indexOf(s)>-1)
          fileList.add(((FileElement)elemento).path
                                    +"/"+ el.toString());
       }
       else{
          String el = ((DirectoryElement)elemento).name;
          if (el.indexOf(s)>-1)
             fileList.add(((DirectoryElement)
                            elemento).path + el.toString());
             fileList.addAll(((DirectoryElement)
                            elemento).searchByName(s));
       }
    }
```

Facile e veloce, proprio come ci prometteva Javolution!

CONCLUSIONI

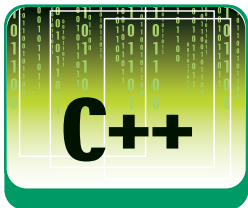
Javolution si è rivelato un package molto interessante non solo in ambito realtime, ma in qualsiasi applicazione che per un motivo od un altro abbia bisogno di una gestione della memoria più curata. Soli 200 kb di file jar lo rendono appetibile addirittura in ambito microedition, soprattutto per via del supporto alla reflection o per le funzioni matematiche. Vivamente consigliato.

Luca Mattei

L'AUTORE

Luca Mattei è laureando in ingegneria informatica, lavora come progettista software per una Mobile Company che offre la sua consulenza ai gestori di telefonia e alle major del settore ICT.
È uno degli amministratori di www.JavaStaff.com**, portale dedicato al mondo Java. Potete contattarlo scrivendo a:** luca.mattei@javastaff.com

Accesso universale ai database con C++

Parleremo di Oracle, ODBC and DB2 Template Library, per gli amici OTL. Una libreria leggerissima, dotata di alcune particolarità estremamente interessanti e che ci svincola totalmente dal database usato

L'idea è molto semplice, al contempo la sua concretizzazione in una forma perfetta viene inseguita ormai da molti anni.
Si tratta di creare una libreria che offra al programmatore un'unica interfaccia per la gestione di più formati di database. Questo tipo di astrazione consente ovviamente di usare un solo codice e svincola il programmatore dal dover programmare un "driver" per i vari tipi di database esistenti. In questo articolo ci occuperemo di *OTL: Oracle, Odbc & DB2-CLI Template Library,* una libreria OpenSource sviluppata da Sergei Kuchin che offre una serie di caratteristiche piuttosto interessanti. Prima di tutto, l'intero codice è contenuto in un header di appena 500k, in questa manciata di byte c'è tutto il necessario per supportare Oracle 7, Oracle 8, Oracle 8i, Oracle 9i, Oracle 10g, DB2, ODBC 3.x, ODBC 2.5 (Oracle, MS SQL Server, Sybase, MySQL, DB2, Interbase/Firebird, PostgreSQL, SQLite, etc.). Se qualcosa non è presente in questa lista, molto probabilmente lo sarà in breve tempo, di fatto l'elenco dei DB supportati è in piena crescita. Dal punto di vista strettamente programmatico una delle carte vincenti della libreria è costituito dall'uso degli Stream molto simili a quelli usati in C++ con il comune operatore >>. Durante la compilazione il codice OTL viene espanso in linea direttamente all'interno del programma utente e trasformato nelle chiamate dirette delle API, in tal modo si assicurano performance, affidabilità e sicurezza ai thread negli ambienti multiprocessori. È scritto in C++ ANSI ed si integra perfettamente nel modello di programmazione dettato dalla *Standard Template Library*, attraverso *stream_iterator* compatibili STL e accetta l'uso delle stringhe standard. La licenza di distribuzione è semplice e liberale, sulla falsariga della nuova BSD, quindi OTL può essere usata sia in programmi liberi che in software proprietario.

HEADER E COMPILAZIONE

Poiché la libreria è staticamente inclusa nel codice utente come un file *include,* tutta la configurazione avviene attraverso la definizione di valori con la direttiva #define del preprocessore. L'unica *#define* necessaria è quella che indica il tipo di database a cui accedere. Ad esempio,

```
#define OTL_DB2_CLI
```

permette la corretta compilazione del codice relativo alla Command Level Interface (l'interfaccia API) del DB2, invece la definizione alternativa,

```
#define OTL_ODBC
```

si riferisce all'uso delle primitive relative ad ODBC. In Unix, va notato, l'uso dell'ODBC è realizzato attraverso i driver di database specifici, attivati dalla precedente #define, ma spesso anche da una libreria ponte verso il sistema operativo (tradizionalmente il progetto libero unixODBC) che deve quindi essere attivata esplicitamente con l'uso della seguente dichiarazione:

```
#define OTL_ODBC_UNIX
```

Queste sono le uniche definizioni necessarie per poter usare la OTL e devono sempre precedere l'inclusione dell'header:

```
#include <otlv4.h>
```

COME INIZIARE

È necessario scaricare il file *otlv4_h .zip* dal sito http://otl.sourceforge.net/ oppure utilizzare la versione contenuta in questo stesso numero di ioProgrammo. All'interno di questo ZIP è contenuto un unico file: *otlv4.h,* che va copiato nella stessa directory che conterrà i sorgenti del nostro programma. Tutti gli esempi dell'articolo sono stati sviluppati in ambiente Linux; il compilatore utilizzato è uno GNU C++.
Affinché tutto funzionasse, abbiamo installato i pacchetti *UnixODBC* e *UnixODBCdev*.
In ambiente Windows potrete tranquillamente utilizzare Dev C++.
La libreria funziona tranquillamente sia in ambiente Linux sia in ambiente Windows.

La mia sezione di include è la seguente

```
#include <iostream>
using namespace std;
#define OTL_ODBC_MYSQL      // driver odbc di MYSQL
#define OTL_ODBC_UNIX    // adattatore ODBC
#include "otlv4.h"
```

COME COMPILARE

Se non si vogliono utilizzare i define si può anche delegare l'uso della libreria alla linea di comando del compilatore, usando la definizione dei simboli del preprocessore attraverso l'opzione *-D*, ad esempio:

```
gcc -DOTL_ODBC_MYSQL -DOTL_ODBC_UNIX
                              -o test test.cc [...]
```

Esistono altre opzioni che possono essere definite allo stesso modo per ottenere comportamenti specifici dalla libreria. Sono cose molto particolari, spesso dipendenti dal database scelto o aggiuntive.
Ad esempio:

```
#define OTL_BIG_INT
```

permette il supporto degli interi lunghi doppi a 64 bit con segno non presenti nello standard ANSI C++ (*_int64* nel VC++, o *long long* in gcc), che però non sono supportati nei DB Oracle. L'elenco completo delle opzioni attivabili è disponibile nella documentazione del prodotto.
La OTL non ha una libreria di supporto da collegare in fase di compilazione ma è invece necessario procedere al collegamento delle librerie del supporto del database.
Così, ad esempio nel caso del DB2 di IBM, sarà necessario usare una linea di comando per la compilazione come la seguente:

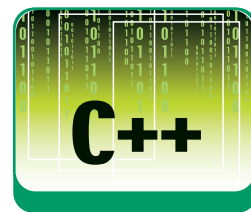
```
gcc -DOTL_DB2_CLI -I/home/db2/sqllib/include
      -o test test.cc -lstdc++ -L/home/db2/sqllib/lib -ldb2
```

oppure per la compilazione dello stesso programma usando l'ODBC specifico per MySQL sotto Linux (con tutte le librerie installate nelle directory standard):

```
gcc -DOTL_ODBC_MYSQL -DOTL_ODBC_UNIX
                              -o test test.cc -stdc++ -lodbc
```

Come ulteriore esempio si può pensare al porting del programma di test sotto Mac OS X (quindi un'architettura BSD) usando il driver iODBC cone la seguente linea di comando:

```
gcc -DOTL_IODBC_BSD-o test test.cc -stdc++ -liodbc
```

Una volta agganciate le librerie, tutti i riferimenti introdotti da OTL saranno correttamente risolti in fase di collegamento.

MODELLO DI PROGRAMMAZIONE

OTL rappresenta una evoluzione interessante rispetto alle interfacce native dei database, soprattutto per la mantenuta promessa di poter sviluppare codice multi-piattaforma per differenti database senza ricorrere a noiose sezioni alternative di compilazione. I passi necessari per realizzare una sessione completa di interazione con un database sono i seguenti:

- *Inizializzazione (otl_initialize)*
- *Connessione (db.rlogon)*
- *Esecuzione Statament SQL (otl_stream e operatore <<)*
- *Acquisizione dei risultati (operatore >>)*
- *Disconnessione (db.logoff)*

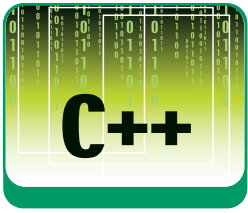
All'inizio del programma sarà necessario attivare il metodo statico *otl_initialize()* della classe *otl_connect* per ottenere l'inizializzazione del sottosistema di gestione del collegamento alla base di dati. È possibile attivare il supporto multi-threaded indicando esplicitamente il parametro opzionale *threaded_mode* di questa funzione al valore 1. Successivamente si potranno definire ed usare gli oggetti appartenenti alla classe *otl_connect* (db nei seguenti esempi)

```
otl_connect::otl_initialize();
otl_connect db;
```

Il passo successivo è il collegamento al database attraverso la funzione *rlogon()*

```
db.rlogon(<stringa di connessione>);
```

La stringa di connessione può essere composta secondo differenti modalità, indipendentemente dal database effettivamente connesso:

- **<user>/<password>** ad esempio *"exedre/pippo"* come per le connessioni a database Oracle locali;
- **<user>/<password>@<conn>** dove *<conn>* può essere l'*ALIAS_TNS* di Oracle o il DSN dell'ODBC, ad esempio *"exedre/pippo@calvin"* usando la sintassi adottata per le connessioni a database Oracle remoti;
- **DSN=<dsn>;UID=<user>;PWD=<password>** ad esempio *"DSN=calvin;UID=exedre;PWD=pippo"* come per le connessioni ODBC o DB2;

La disconnessione da un database si ha con:

```
db.logoff();
```

L'esecuzione di un comando SQL completamente determinato nella stringa di definizione (in gergo detto 'SQL statico') si può semplicemente ottenere usando il membro statico *direct_exec* della classe *otl_cursor*, ad esempio:

```
otl_cursor::direct_exec(db, "drop table test");
otl_cursor::direct_exec(db, "insert into tab1 from
                              tab2 where field = 1 ");
```

Il mio main è il seguente:

```
otl_connect db;
int main()
{  otl_connect::otl_initialize();
    try{
      db.rlogon("UID=root;PWD="";DSN=myodbc");
      otl_cursor::direct_exec(db,"drop
                table test_tab",otl_exception::disabled);
      otl_cursor::direct_exec(db,"create table
                     test_tab(f1 int, f2 varchar(30))");
      insert(); }
   catch(otl_exception& p){
      cerr<<p.msg<<endl; // stampa un eventuale errore
      cerr<<p.stm_text<<endl; // stampa la query SQL
                                   che ha generato l'errore
      cerr<<p.var_info<<endl; // stampa la variabile che
                                      ha causato l'errore }
   db.logoff();
   return 0;
}
```

Ho dichiarato l'oggetto *db* al di fuori del main per una questione di comodità, lo riutilizzerò infatti nella funzione *insert();*

UNO STREAM PER IL DB

Il concetto di *stream* è sufficientemente noto. Esso è un flusso, una sorta di tubo in cui un dato entra da un'estremità ed esce dall'estremità opposta, probabilmente trasformato in qualche sua componente. Una Query SQL non è molto diversa da uno stream, c'è un vettore di dati in input costituito ad esempio nel caso della select dalle colonne da cui reperire i dati e da qualche parametro della clausola *Where* e c'è un flusso di dati in output anche esso il più delle volte rappresentabile in formato vettoriale. Il buffer di input è rappresentato dalle variabili di input del comando mentre quello di output dai record del risultato suddivisi in campi che saranno singolarmente riversati nelle variabili. Come tutti gli stream in C++ anche un oggetto appartenente alla classe *otl_stream* viene manipolato con gli operatori di *shift* >> e <<. Così per introdurre le variabili di input si può usare un'espressione come la successiva:

```
db << in1 << in2 << ... << inN;
```

mentre quella seguente rappresenta l'estrazione di variabili di output dallo stream dopo l'esecuzione del comando indicato nel costruttore:

```
db >> out1 >> out2 >> ... >> outN;
```

Il concetto di *stream* sarà estremamente chiaro considerando questo spezzone di codice:

```
void insert()
{  otl_stream o(1,"insert into test_tab
                   values(:f1<int>,:f2<char[31]>)", db);
 char f2;
 for(int i=1;i<=100;++i){
   f2='*';
   o<<i<<f2;}
}
```

NOTA

ODBC UNIX

Il file di configurazione per settare un driver ODBC è */etc/odbc.ini*

COMANDI CON VARIABILI COLLEGATE

Lo stream OTL include un piccolo parser delle stringhe SQL che riconosce l'uso di variabili di binding la cui sintassi è particolarmente semplice e completa. In questo modo un comando SQL, una stored procedure o un blocco PL/SQL, possono contenere indicatori per riferirsi a variabili di input lette direttamente dal programma in shift successivi verso lo stream. La sintassi degli indicatori accettata dal parser OTL è quella posizionale con i punti interrogativi accettata da ODBC o IBM DB2, ad esempio:

```
otl_stream i(10,"INSERT INTO tableVALUE(?,?,?)",db);
```

è anche possibile utilizzare una notazione posizionale e numerata supportata nelle precedenti versioni della OTL, ad esempio:

```
otl_stream i(10,"INSERT INTO table "
"VALUE(:1<int>,:2<char[12],:3<double>)",db);
```

O anche una nuova notazione, posizionale ma denominata, come nel seguente esempio:

```
otl_stream i(10,"INSERT INTO table"
"VALUE
(:ID<int>,:nome<char[12]>,:salario<double>)",db);
```

Quest'ultima modalità è stata dedotta dalla notazione utilizzata in Oracle ed estesa con la definizione del tipo di dato. Ciò permette allo stream di prede-

terminare la struttura delle variabili del buffer e solleva il programmatore dall'incomodo di dover dichiarare un vettore del tipo corretto per ogni colonna del risultato. È quindi possibile lavorare solo con le relative variabili scalari lasciando allo stream il compito di allocare i buffer opportuni. L'interazione con lo stream avviene attraverso il caricamento delle variabili di input, che in tutti e tre gli esempi precedenti è semplicemente:

```
i << 18 << "Thorstein Veblen" << 1000000.00 ;
```

Un differente esempio è rappresentato dalla seguente esecuzione di una *SELECT* dinamica:

```
otl_stream i(10,"SELECT name, salario "
"FROM table WHERE eta=:eta<int>",db);
vector<double> stot = 0.0;
string name;
double sal;
for(int e=0;e<100;e++) {
        i << e ; // carica l'età
        while (!i.eof()) {
          i >> name >> sal ;
          stot[e] += sal;}
}
```

sarebbe stato più intelligente utilizzare la funzione SUM() dell'SQL per raggiungere più facilmente lo stesso risultato, ma il nostro intento era ovviamente puramente didattico. Sebbene il funzionamento di uno stream con variabili di input collegate sia completamente automatico è fondamentale che tutte le variabili di input siano effettivamente specificate. Non basta caricarne solo una parte anche se l'operatore di *shift (<<)* permette un caricamento solo parziale come ad esempio in:

```
i << nome ;
i << cognome ;
```

Solo il caricamento completo di tutte le variabili di input scatena gli eventi che portano all'esecuzione della query e quindi ad ottenere il risultato.

```
i >> risultato ;
```

Nel caso in cui non tutte le variabili siano state caricate una eventuale operazione di shift dallo stream provocherà una eccezione, che se non gestita bloccherà l'intero programma.

GESTIONE DEGLI ERRORI

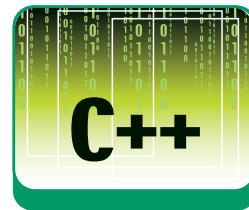
La libreria basa la comunicazione di eventuali condizioni d'errore sull'uso delle eccezioni del linguaggio. Anche questo rende OTL più vicino alla sensibilità dei moderni programmatori C++. Nella realizzazione di programmi reali è quindi buona norma racchiudere le istruzioni di accesso al database in sezioni critiche (blocchi *try/catch*) in cui valutare la possibilità di ricevere eccezioni come risultato. La libreria OTL mette a disposizione una classe di comunicazione delle eccezioni denominata *otl_exception* che ha a disposizione alcuni campi pubblici in cui sono riportate informazioni relative alle condizioni d'errore tra cui i primi 2048 caratteri della query SQL che ha generato l'errore *(char stm_text [2048])*, informazioni sulla variabile incompatibile con l'operazione richiesta *(char var_info [256])*, il messaggio d'errore di una eccezione definita all'interno di OTL *(unsigned char msg [1000])*, il relativo codice d'errore *(int code)* e lo stato SQL così come comunicato da ODBC o DB2-CLI *(unsigned char sqlstate [1000])* che per Oracle OCI è vuoto. Le eccezioni possono anche essere disabilitate istruzione per istruzione, cosa molto utile quando si vogliono dare comandi opzionali che possono fallire senza per questo compromettere il programma. Un esempio d'uso è il seguente:

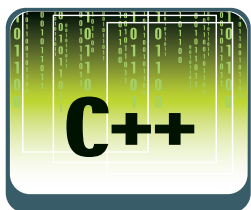
```
int main() {
otl_connect::otl_initialize();
try{
db.rlogon("exedre/pippo@calvin");
otl_cursor::direct_exec ( db, "drop table test",
                                otl_exception::disabled );
otl_cursor::direct_exec ( db, "create table test
                         (f1 number, f2 varchar2(30))" );
[...] }
catch(otl_exception& p){
cerr<<"MSG: "<<p.msg<<endl;
cerr<<"SQL: "<<p.stm_text<<endl;
cerr<<"VAR: "<<p.var_info<<endl; }
db.logoff();
return 0;
}
```

Eventuali eccezioni non catturate in blocchi *try* si propagano secondo le regole del C++ e se non gestite al livello più alto del programma, nella main cioè, causano il *core dump* dell'intera applicazione.

CONCLUSIONI

La Oracle, ODBC & DB2-CLI Template Library semplifica in modo efficace la programmazione di database in C++ attraverso l'uso di costrutti basati sulla stessa logica di quelli standard. È chiara, diretta, ben progettata e non ha orpelli inutili. Raggiunge ampiamente i due risultati che si prefiggeva: permettere la scrittura di un codice C++ più pulito e permettere il supporto trasparente dei database ODBC o nativi.

Emmanuele Somma

Effetti 3D con il Normal Mapping

Il vostro Video gioco è piatto? Non dà più le emozioni di un tempo? Proiettiamolo in un'altra dimensione usando una tecnica che ci consenta di dare profondità alle immagini

Le moderne tecniche di rappresentazione di una scena 3D real-time, consentono di disegnare a schermo oggetti apparentemente molto complessi, ma dalla geometria relativamente semplice. Questi effetti sono ottenuti in diversi modi. Ad esempio utilizzando i Pixel e/o Vertex Shader, di cui già abbiamo parlato su queste pagine. Il concetto fondamentale che sottende all'utilizzo di queste tecniche è la manipolazione dei singoli pixel in base a informazioni quali intensità della luce, incidenza del vettore luminoso, geometria della mesh su cui il pixel risiede ecc. Una tecnica molto efficace e di grande impatto visivo è quella che prevede l'illuminazione dinamica della scena, attraverso l'utilizzo di particolari mappe, dette *Normal Map* (NM, mappe di normali).

LA TECNICA

Le NM altro non sono che immagini, proprio come le comunissime texture. Analogamente alle texture, inoltre, sono applicate a superfici 3D secondo delle determinate coordinate di mapping (si parla infatti di normal mapping così come di texture mapping). La differenza tra texture e NM sta nell'informazione associata a ciascun "pixel". Nelle texture un pixel è composto da 3 valori che rappresentano le componenti RGB (rosso-verde-blu) del pixel stesso e ne determinano il colore. Nelle NM questi 3 valori non rappresentano componenti cromatiche, bensì direzionali. Sono le componenti *X*, *Y* e *Z* della normale alla superficie 3D nel punto in cui risiede il pixel. In altre parole, si ha il vettore che determina la direzione in cui il pixel "guarda". Questo consente di disegnare dettagli in maniera molto più "fine". Una NM adeguatamente costruita può "ingannare" l'occhio e fargli percepire una geometria 3D molto più complessa di quella che

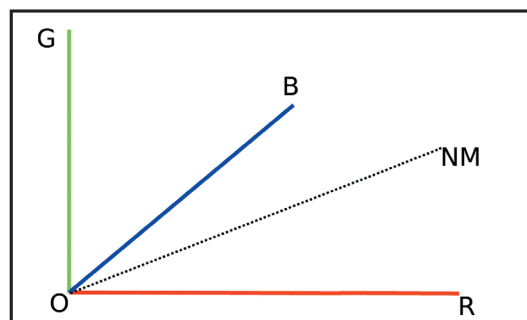



Fig. 1: Una rappresentazione geometrica di cosa si potrebbe intendere per Normal Map

è in realtà. In teoria una NM potrebbe fare apparire un cubo come se fosse a tutti gli effetti una sfera. Solo il contorno della figura rimarrebbe quadrato, anziché circolare. Un po' strano da immaginare, ma è proprio così. Supponiamo che il pixel di cui stiamo parlando sia l'origine degli assi RGB, come in figura. Un vettore 0 0,0,100 ovvero 100% di rosso rappresenta una normale completamente spostata verso destra, lo 0% di rosso una normale completamente a sinistra mentre con il 50% la normale è in posizione perpendicolare alla superficie, l'1% di rosso una normale completamente a sinistra mentre con il 50% la normale è in posizione perpendicolare alla superficie. Ovviamente questo si applica anche alle componenti verde e blu con le coordinate *Y* e *Z*. Per la componente *blu-Z* non sono contemplati valori al di sotto del 50% in quanto significherebbero una normale diretta verso l'interno della superficie, il

Fig. 2: In questa foto abbiamo applicato una normal mapping calcolando i valori RGB dei pixel e disegnando la normale spostata verso il rosso

che non avrebbe senso. Un esempio di rappresentazione 3D della normale calcolata in RGB è visibile in **Figura 1.** Nella **Figura 2** è possibile vedere un classico esempio di NM, in cui i valori *XYZ* vengono visualizzati come se fossero RGB. La texture relativa alla stessa immagine è quella di **Figura 3**.

Fig. 3: **In questa foto si vede la texture originale priva dell'applicazione di effetti di normal mapping**

IMPLEMENTAZIONE

Le NM sono implementabili, e supportate, da entrambe le principali librerie 3D presenti sulla scena: Direct3D e OpenGL. Programmare tutto il codice relativo alle NM con una di queste librerie, tuttavia, ci devierebbe dallo scopo didattico del nostro discorso, a causa della notevole mole di codice necessaria ma non strettamente legata a questo argomento. Per concentrarci quindi sull'applicazione della tecnica in senso stretto, faremo ricorso al motore grafico IrrLicht (*http://irrlicht.sourceforge.net*), di cui abbiamo già ampiamente parlato nei numeri precedenti. IrrLicht consente di applicare in maniera indolore una NM a una mesh caricata nella scena.
Tra le funzionalità offerte da questo motore c'è anche quella di creare una NM a partire da un altro tipo di mappa, la *Height Map* (HM, mappa di altezze). La HM conserva le informazioni relative all'altezza (anziché alla normale) di ciascun pixel ed è un metodo meno raffinato di ottenere un risultato simile a quello fornito dalle NM. Come si vede è una immagine in toni di grigio: più è chiaro il pixel, maggiore sarà la sua altezza sulla superficie.
Il percorso che seguiremo nel codice di esempio che presenteremo è questo:

- Carichiamo la geometria di una mesh sferica rappresentante il pianeta Terra da un file *.x*.
- Impostiamo come texture di livello 0 la texture colorata, con gli oceani i continenti ecc. (**Figura 4**).

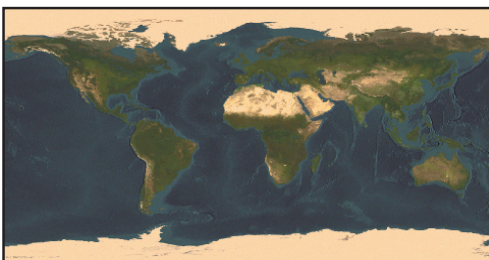



Fig. 4: **La nostra texture di livello 0 sarà una normalissima texture piatta**

- Carichiamo la HM da file (**Figura 5**).

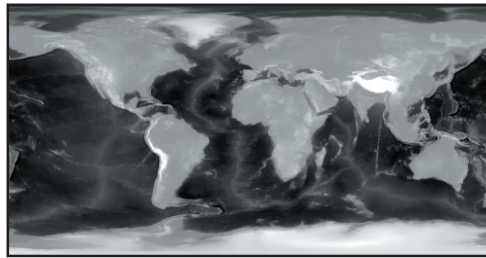



Fig. 5: **La Horizzontal Map apparirà in bianco e nero ma è già visibile un minimo di effetto 3D**

- Impostiamo come texture di livello 1 la NM ottenuta dalla HM di prima.
- Creiamo un altro oggetto identico, ma senza NM, per mostrare la differenza.

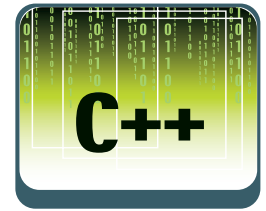
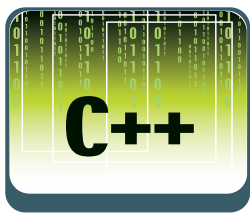
IL CODICE

L'inizializzazione di un programma IrrLicht segue lo schema classico già discusso in passato:

```
// Creo il device utilizzando le DirectX 9
video::E_DRIVER_TYPE driverType =
                         video::EDT_DIRECTX9;
IrrlichtDevice* device = createDevice
(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
return 1;
// Ottengo i puntatori a driver, scene manager e
                                  environment
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->
                            getSceneManager();
gui::IGUIEnvironment* env = device->
                          getGUIEnvironment();
```

La cosa importante quando si tratta di NM è assicurarsi che esse abbiamo il necessario numero di bit (32). Infatti saranno utilizzati 8 bit per ciascuna componente *(XYZ)*, per un totale di 24 bit. Questo vuol dire che le texture a 16 bit non sono sufficienti. Impostiamo dunque IrrLicht per la creazione da file grafici di texture a 32 bit:

```
// Imposto le texture sui 32 bit
driver->setTextureCreationFlag(
              video::ETCF_ALWAYS_32_BIT, true);
```

Aggiungiamo un nodo "videocamera" che ci consentirà di spostarci all'interno della scena muovendo il mouse o utilizzando le frecce direzionali sulla tastiera. In questo modo potremo apprezzare il risultato da diverse angolazioni. Eliminiamo inoltre il cursore del mouse.

EDITOR 3D
Un buon Editor 3D con supporto ai file .x è Blender:
http://www.blender.it

```
// Aggiungo la videocamera
scene::ICameraSceneNode* camera =
smgr->addCameraSceneNodeFPS(0,100.0f,300.0f);
camera->setPosition(core::vector3df(0,0,-150));
// Nascondo il cursore del mouse
device->getCursorControl()->setVisible(false);
```

Creiamo a questo punto i nodi della scena rappresentanti gli oggetti *"pianeta Terra"*, che visualizzeremo uno accanto all'altro. Per farlo carichiamo la geometria di una sfera dal file *"earth.x"*. Questo file contiene anche le coordinate di mapping che verranno utilizzate per texture e NM.

```
// Aggiungo due nodi creati dalla mesh earth.x
scene::IAnimatedMesh* earthMesh = smgr->
                              getMesh("earth.x");
if (earthMesh) {
  // creo una copia della mesh calcolando le tangenti
  scene::IMesh* tangentSphereMesh = smgr->
      getMeshManipulator()->createMeshWithTangents(
                           earthMesh->getMesh(0));
  // Ingrandisco la mesh di un fattore 50
  smgr->getMeshManipulator()->scaleMesh(
    tangentSphereMesh, core::vector3df(50,50,50));
  // Creo 2 nodi dalla stessa mesh e li posiziono
  // uno accanto all'altro
  scene::ISceneNode* sphere1 = smgr->
          addMeshSceneNode(tangentSphereMesh);
      sphere1->setPosition(core::vector3df(70,0,0));
  scene::ISceneNode* sphere2 = smgr->
          addMeshSceneNode(tangentSphereMesh);
  sphere2->setPosition(core::vector3df(-70,0,0));
```

A questo punto possiamo creare la NM. Per farlo, come già accennato, utilizziamo la funzione *makeNormalMapTexture()* di IrrLicht. Questa funzione crea una NM a partire da una HM:

```
  // Carico la height map e creo da essa una normal map
  video::ITexture* earthNormalMap =
              driver->getTexture("earthbump.bmp");
  driver->makeNormalMapTexture(
                      earthNormalMap, 10.0f);
```

Ora la geometria è caricata ma la HM ancora non è stata utilizzata. Se visualizzassimo adesso il risultato vedremmo due sfere dai colori piatti, poiché ancora non è stata settata la NM. Settiamola quindi con le seguenti righe:

```
// Imposto la texture e il tipo di materiale
sphere1->setMaterialTexture(1, earthNormalMap);
sphere1->setMaterialType
(video::EMT_NORMAL_MAP_TRANSPARENT_VERTEX_
                                   ALPHA);
```

Concludiamo questo blocco di istruzioni eliminando l'oggetto contenente la geometria della sfera, che non ci servirà più:

```
// drop della mesh
tangentSphereMesh->drop();
}
```

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

E LUCE FU!

I notevoli risultati visivi consentiti dalle NM sono apprezzabili maggiormente con oggetti in movimento, le immagini statiche non rendono giustizia. Inoltre un fattore fondamentale è il giusto posizionamento di fonti luminose. Aggiungiamo quindi due nodi contenenti delle luci di colore bianco.

```
// Aggiungo 2 luci
scene::ILightSceneNode* light1 = smgr->
      addLightSceneNode(0, core::vector3df(0,0,0),
     video::SColorf(1.0f, 1.0f, 1.0f, 0.0f), 1000.0f);
scene::ILightSceneNode* light2 = smgr->
      addLightSceneNode(0, core::vector3df(0,0,0),
     video::SColorf(1.0f, 1.0f, 1.0f, 0.0f), 1000.0f);
```

Per un maggiore effetto visivo aggiungiamo anche un movimento circolare ai nodi contenenti le luci:

```
// Imposto un movimento circolare in senso opposto
// per le 2 luci
scene::ISceneNodeAnimator* anim1 = smgr->
          createFlyCircleAnimator (core::vector3df(
                      0,300,0),1000.0f, -0.002f);
light1->addAnimator(anim1);
anim1->drop();
scene::ISceneNodeAnimator* anim2 = smgr->
          createFlyCircleAnimator (core::vector3df(
                      0,-300,0),1000.0f, 0.002f);
light2->addAnimator(anim2);
anim2->drop();
```

Per finire scriviamo il ciclo principale del programma che fa uso di IrrLicht. Tra le chiamate a *beginScene()* e *endScene()* invochiamo unicamente il metodo *drawAll()* dello SceneManager, che si occuperà di disegnare la scena 3D applicando gli effetti grafici inseriti in precedenza:

```
// Ciclo principale del programma
while(device->run())
if (device->isWindowActive()) {
  driver->beginScene(true, true, 0);
         smgr->drawAll();
         driver->endScene(); }
device->drop();
```

Il risultato finale è visibile in **Figura 6**. Sulla sinistra è visibile la mesh senza la NM, che ha un

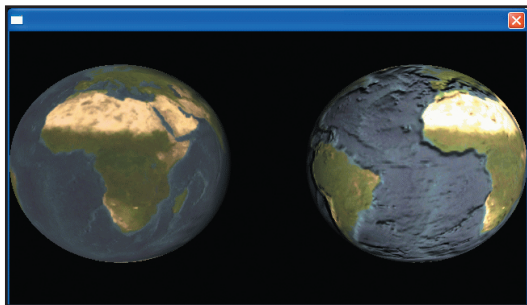
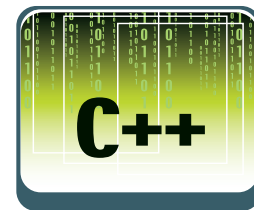



Fig. 6: ***A sinistra l'immagine prima di avere applicato la normal map a destra il risultato 3d con l'applicazione della NM***

aspetto piuttosto "liscio". Sulla destra possiamo notare la superficie più "crespa" data dall'applicazione della NM.

CONCLUSIONI

Le moderne schede video 3D hanno la potenza sufficiente a disegnare in real-time scene in cui l'illuminazione viene calcolata sulla base di informazioni relative ad ogni singolo pixel. Una delle tecniche che permette di fare questo è quella che fa uso delle Normal Map. In questo articolo abbiamo spiegato come utilizzare le NM in maniera semplice, sfruttando il motore grafico open source e gratuito IrrLicht.
Abbiamo applicato la tecnica facendo uso anche di Height Map e confrontando la differenza fra una mesh con una semplice texture e la stessa mesh con la NM applicata.

Fabio Alfredo Marroccelli

CREARE UNA HEIGHT MAP

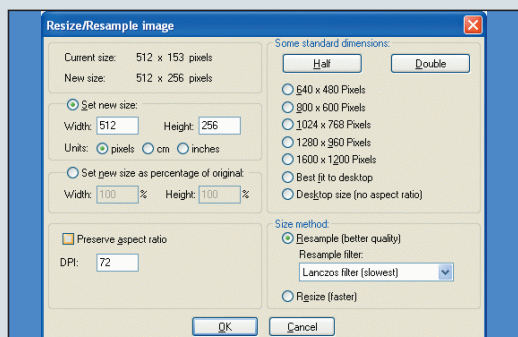
Realizzare un HM è abbastanza semplice. Sarà sufficiente avere a disposizione un qualunque programma di ritocco fotografico 2d ed il gioco è fatto, vediamo come procedere

> GLI STRUMENTI

1 Qualsiasi immagine dai colori forti e i contorni ben definiti può essere utilizzata per creare una HM. Quello di cui si ha bisogno è un programma (anche semplice) di grafica.

> DEFINIAMO I COLORI

2 La prima cosa da fare è portare lo spazio cromatico dell'immagine a 256 colori e in toni di grigio. Le parti dell'immagine più chiare saranno quelle maggiormente in rilievo.

> LE DIMENSIONI

3 Ridimensioniamo l'immagine in modo che le dimensioni finali siano 512x256. Questo ci permetterà di utilizzare la HM col programma sviluppato nell'articolo.

> IL RISULTATO FINALE

4 Salviamo la HM in formato bitmap e rinominiamola in *"earthbump.bmp"*. Copiamola nella cartella in cui è presente il file *"earth.x"* e eseguiamo il nostro programma.

Persistenza dei dati con pBeans

Quando Hibernate si rivela troppo complesso e Castor non è sufficiente, arriva Pbeans a salvarvi. Un tool leggero, semplice da usare, che non richiede attenzione ai database sottostanti

Uno dei principali e più macchinosi problemi nello sviluppo della maggior parte delle applicazioni con linguaggi object oriented è sicuramente la gestione della persistenza degli oggetti su database. Tale questione, trasversale a molte applicazioni, non è affatto di banale risoluzione. Molte sono le librerie, anche open source, che gestiscono tale aspetto. Quella che attualmente sembra ricoprire il ruolo di leader è Hibernate, molto flessibile e con elevate prestazioni, ma che necessita di uno studio approfondito per poterne sfruttare appieno le caratteristiche. Altre soluzioni sono Cayenne, Castor, la specifica JDO con numerose implementazioni sia proprietarie sia open source, e la specifica J2EE che prevede la persistenza degli entity bean *CMP (Container Manage Persistence)*. In questo articolo utilizzeremo la libreria *pBeans* per la gestione della persistenza di classi Java. *pBeans* si distingue dalle altre librerie per l'estrema semplicità di utilizzo e anche se non raggiunge la flessibilità di Hibernate, non ne eguaglia neanche la complessità di configurazione. *pBeans* è in grado di funzionare con svariati database: MySQL, Microsoft SQL Server e anche HSQLDB che utilizzeremo nell'esempio.
Ciò che *pBeans* fa è sostanzialmente salvare su una tabella lo stato di una classe *Java* e ricostruire da un record un oggetto.

PRIMI PASSI

Innanzitutto sarà necessario creare una directory che conterrà tutti i file del progetto. All'interno di questa cartella, dovranno essere create delle ulteriori sottodirectory fino a raggiungere la seguente struttura

- **src/:** conterrà tutti i sorgenti java.
- **build/:** conterrà i *class* compilati dai sorgenti java
- **lib/:** conterrà i file *.jar* contenenti le librerie necessarie allo sviluppo

Procuriamoci ora *pBeans* e HSQLDB. La home page di pBeans si trova all'indirizzo *http://pBeans.sourceforge.net/*. Cliccate su *"download"* e succesivamente su *"sourceforge download area"* e dalla pagina di download scaricate il file *pBeans_1_3_1.zip*. Scompattate il file, individuate la cartella *"lib"* e copiate il file *"pBeans.jar"* nella cartella *"lib"* del progetto. Questa libreria contiene tutti i class file necessari all'utilizzo di pBeans. Per installare HSQLDB è necessario scaricare il database da Sourceforge. Collegatevi all'indirizzo *http://hsqldb.sourceforge .net/*, e seguite il link *"download"* che vi porterà alla solita pagina. Scaricate l'ultima versione stabile che al momento della stesura dell'articolo è la 1.7.3, contenuta nel file *hsqldb_1_7_3_3.zip*. Scompattate il file nella directory del progetto. Questo dovrebbe creare una nuova cartella */hsqldb*. Anche in questo caso individuate la cartella *"lib"* e copiate il file *"hsqldb.jar"* nella cartella *"lib"* del progetto. Ovviamente trovate tutto il necessario anche sul CD di ioProgrammo.

LANCIARE IL SERVER HSQLDB

Portatevi nella cartella */hsqldb* e digitate la seguente riga di comando per avviare il server.

```
java -cp lib/hsqldb.jar org.hsqldb.Server -database
     .0books -dbname.0 books -silent false -trace true
```

Viene lanciato il server con un database di nome *"books"* con la massima *"verbosità"* possibile, in modo che sia chiaro dai messaggi che appariranno sulla console le chiamate SQL operate da pBeans. Per *"Verbosità"* intendiamo una forzatura fatta al termine inglese *"verbose"* che indica il quantitativo di messaggi mostrati a video da un software e che ne

indicano lo stato di funzionamento. Con l'opzione *-silent* facciamo in modo che HSQLDB ci restituisca a video un numero elevato di messaggi che ci aiuteranno a capire se e come il *db* sta funzionando.
Se l'avvio del database procede correttamente dovrebbero apparire qualcosa di simile.

```
[Server@1820dda]: Startup sequence completed in
                                        951 ms.
[Server@1820dda]: 2005-04-08 23:17:55.425
                          HSQLDB server 1.7.3 is online
[Server@1820dda]: To close normally, connect and
                               execute SHUTDOWN SQL
[Server@1820dda]: From command line, use
                          [Ctrl]+[C] to abort abruptly
```

Per verificare l'installazione di HSQLDB lanciamo da console il seguente comando.

```
java -cp lib/hsqldb.jar org.hsqldb.util
                              .DatabaseManagerSwing
```

A questo punto nel popup immettiamo i dati per connetterci al database creato.

- **Type:** *HSQL Database Engine Server*
- **Driver:** *org.hsqldb.jdbcDriver*
- **Url:** *jdbc:hsqldb:hsql://localhost/books*
- **User:** *sa*
- *Lasciare vuoto il campo password*

SCRIVERE UNA CLASSE PERSISTENTE

Una classe persistente per pBeans deve realizzare l'interfaccia *Persistent* di pBeans. Questa interfaccia che non impone l'implementazione di alcun metodo è una *"tag interface"* che ha esclusivamente il compito di marcare una caratteristica di una classe, in questo caso la sua persistenza attraverso pBeans. Sostanzialmente rendere persistente un oggetto significa salvarne lo stato su database. Se la classe *"Book"* ha una proprietà persistente, diciamo *"title"*, di tipo *String*, sarà necessario dotare la classe di un *setter* e di un *getter* relativi a questa proprietà:

- *void setTitle(String t)*
- *String getTitle()*

Nella specifica JavaBeans questa coppia di metodi sottintende l'esistenza di una proprietà *"Title"* che pBeans renderà persistente. pBeans non rende tuttavia persistenti proprietà di qualsiasi tipo ma esclusivamente quelle di tipo primitivo o istanza di classi *boxing (Integer, Long,* ecc...*), String, java.util.Date, java.sql.Date,* qualsiasi altra classe *Persistent* e *PersistendID*, una classe *pBeans* che rappresenta l'ID di un oggetto persistente. Nel seguente esempio viene realizzata la classe *"Book"*. Le relative caratteristiche che vogliamo rendere persistenti sono il titolo, l'anno di pubblicazione e il nome della casa editrice. Da ciò deriva la definizione di 3 coppie di metodi *setter* e *getter*. Create il file *Book.java* all'interno della cartella */src/ioProgrammo*.

```
package ioprogrammo;
import net.sourceforge.pBeans.Persistent;
public class Book implements Persistent {
  private String title;
  private String publisher;
  private int year;
  public Book(String title, int year, String publisher){
    this.title = title; this.publisher = publisher;
    this.year = year;}
  //costruttore per specifice JavaBean
  public Book() {this("", 0, "");}
  public String getPublisher() {return publisher;}
  public void setPublisher(String publisher)
                              {this.publisher = publisher;}
  public String getTitle() {return title;}
  public void setTitle(String title) {this.title = title;}
  public int getYear() {return year;}
  public void setYear(int year) {this.year = year;}
  public String toString(){ //descrizione del libro
                                        come String}
}
```

SALVARE UN OGGETTO

Le istanze vengono rese persistenti dalla classe *Store* di PBean. Per creare un'istanza di *Store* è necessario prima di tutto inizializzare un DataSource e poi utilizzarlo nella configurazione dello *Store*. Prima di tutto importiamo tutte le librerie necessarie e dichiariamo uno *Store*.

```
package ioprogrammo;
import java.util.Properties;
import net.sourceforge.pBeans.data.*;
import net.sourceforge.pBeans.util.BeanMapper;
import net.sourceforge.pBeans.*;
import javax.sql.*;
public class Test {
    public static final Store theStore;
```

Nel successivo inizializzatore statico compiliamo una *Properties* con i dati di configurazione. Non è strettamente necessario compilare la *Properties* di configurazione programmaticamente. Sfruttando i metodi della classe *Properties* è facilissimo caricare tali valori da un file di configurazione esterno. Di seguito attraverso la classe helper *BeanMapper* offerta a pBeans trasferiamo la configurazione al DataSource. A questo punto creiamo uno *Store* che lavorerà

Utilizza questo spazio per le tue annotazioni

con il DataSource appena creato. In caso l'istanziazione non dovesse andare a buon fine il programma termina. Lo store è in questo esempio una variabile statica poiché, come indicato nella documentazione di pBeans, è preferibile in un applicazione avere un'unica istanza di *Store* per ogni DataSource, per non incappare in problemi di gestione.

```
{  Properties configuration = new Properties();
   configuration.setPropert("dataSourceClass",
                     "net.sourceforge.pBeans.data
                          .GenericDataSource");
   configuration.setProperty("driverClassName",
                          "org.hsqldb.jdbcDriver");
   configuration.setProperty(
        "url","jdbc:hsqldb:hsql://localhost/books");
   try { Class dataSourceClass = Class.forName(
                       configuration.getProperty(
                             "dataSourceClass"));
     DataSource source = (DataSource)
                  dataSourceClass.newInstance();
     BeanMapper.populateBean(source, configuration);
     theStore = new Store(source);
   } catch (Exception e) {
     e.printStackTrace();
     System.exit(-1); }
}
```

Il metodo *store()* istanzia un paio di libri e cerca di salvarli tramite il metodo *save()*. Il metodo *main* si limita a richiamare il tutto. Il metodo *save()* di *Store* solleva una *"DuplicateEntryException"* nei casi in cui l'oggetto che si desidera salvare sia già presente su DB o se non venga rispettato un vincolo imposto da un indice unique. Tali problemi non si dovrebbero comunque presentare perchè non abbiamo definito alcun indice.

```
  public void store() { Book book1 = new Book
      ("Programmazione Java", 2003, "Edizioni Master");
    Book book2 = new Book("Programmare in
         ambiente Simbian", 2005, "Edizioni Master");
    try { theStore.save(book1);
      theStore.save(book2);
    } catch (StoreException e) {
      e.printStackTrace(); }}
  public static void main(String[] args){
    Test t = new Test();
    t.store(); }
}
```

COMPILARE E LANCIARE L'APPLICAZIONE

Per compilare l'applicazione portatevi nella directory del progetto e dal prompt dei comandi digitate la seguente riga.

```
javac -classpath lib\pBeans.jar -d build
                    src\ioprogrammo\*.java
```

Nel *classpath* non serve includere il *jar* di HSQLDB poiché i driver JDBC vengono caricati a runtime e non vi sono riferimenti a compile time.
Per lanciare il programma digitate invece

```
java -classpath build;lib\pBeans.jar;lib\hsqldb.jar
                       ioprogrammo.Test
```

In questo caso è necessario specificare nel class path il *jar* di HSQLDB. Vediamo dai messaggi di HSQLDB in console le chiamate SQL eseguite da pBeans.
Prima di tutto pBeans crea una tabella in cui tiene traccia per ogni classe di tutti gli identificativi utilizzati, denominati *PersistenId*. Un *PersistentId* non è altro, sostanzialmente, che un numero che identifica univocamente una particolare istanza. La tabella *"TNET_SOURCEFORGE_pBeans_OBJECT"* ha i campi *className* e persistenId. In questa tabella per ogni classe *Persistent Pbean* inserisce tutti i *PersistentId* utilizzati per quella classe. In questo modo nessun *PersistentId* sarà mai assegnato a due oggetti diversi.

```
CREATE TABLE TNET_SOURCEFORGE_pBeans_OBJECT
ALTER TABLE TNET_SOURCEFORGE_pBeans_OBJECT
         CLASS ADD COLUMN className varchar
ALTER TABLE TNET_SOURCEFORGE_pBeans_OBJECT
       CLASS ADD COLUMN persistentID BIGINT
ALTER TABLE TNET_SOURCEFORGE_pBeans_OBJECT
      CLASS ADD COLUMN JP__OBJECTID BIGINT
```

PBean crea poi una tabella che ha un nome derivato dal *full qualified name* della classe *Book*, creando un campo per ogni coppia *setter/getter*, così come si può evincere dalle seguenti invocazioni SQL.

```
CREATE TABLE TIOPROGRAMMO_BOOK
             (JP__OBJECTID BIGINT NOT NULL)
ALTER TABLE TIOPROGRAMMO_BOOK ADD COLUMN
                         publisher varchar
ALTER TABLE TIOPROGRAMMO_BOOK ADD COLUMN
                             title varchar
ALTER TABLE TIOPROGRAMMO_BOOK ADD COLUMN
                              year INTEGER
```

Al momento del salvataggio degli oggetti pBeans esegue delle *insert* nella tabella corrispondente alla classe dell'oggetto da salvare.

```
insert into Tioprogrammo_Book
   (year,title,JP__OBJECTID,publisher) values (?,?,?,?)
insert into Tioprogrammo_Book
   (year,title,JP__OBJECTID,publisher) values (?,?,?,?)
```

Controllando il contenuto delle tabelle possiamo

renderci conto di come ha agito Pbean.

```
JP__OBJECTID     CLASSNAME     PERSISTENTID
------------------- ---------------- -------------------
4708896339122382358 ioprogrammo.Book
                                  6466997603929694911
2180837346892913095 ioprogrammo.Book
                                  7551407953018338209
JP__OBJECTID     PUBLISHER    TITLE              YEAR
------------------- --------- ------------------------- ----
6466997603929694911 Edizioni Master
                    Programmazione Java              2003
7551407953018338209 Edizioni Master
                Programmare in ambiente Simbian 2005
```

OTTENERE LE ISTANZE SALVATE

In generale per eseguire un'operazione riconducibile ad una *SELECT* si utilizzano i metodi *select()* e *selectSingle()* di Store. Per ottenere tutte le istanze di una certa classe rese persistenti si invoca il metodo *select()* di store a cui si passa l'oggetto *Class,* di cui gli oggetti che desideriamo selezionare sono istanze.
Il metodo restituisce un *ResultsIterator* che racchiude tutti i bean persistenti, che può essere facilmente navigato con i relativi metodi *next()* e *hasNext()*.
È anche possibile selezionare solo quei bean che hanno un particolare valore di una proprietà. Per fare ciò è necessario costruire una *Map* che associ una proprietà di un bean al valore che deve avere, e passarla al metodo *select()* di Store; "preleverà" dal DB solo quelle istanze che soddisfano i criteri definiti dalla Map. È ovvio che molta dell'espressività possibile in una clausola *WHERE* in SQL viene persa. Il sistema di definizione dei vincoli infatti impone che il valore della proprietà debba essere proprio quello e che nel caso di due o più vincoli questi debbano essere verificati tutti contemporaneamente. Come è facile immaginare questo si traduce in una clausola *WHERE* del tipo

```
WHERE campo1 = val1 AND campo2 = val2
```

Senza possibilità di variare né l'operatore di confronto, né l'operatore logico. Di seguito il sorgente del metodo da includere nella classe *Test.*

```
  public void select(){
    Map filter = new HashMap();
    filter.put("year", new Integer(2005));
    filter.put("publisher", "Edizioni Master");
    try { ResultsIterator books = theStore.select(
                                   Book.class,filter);
      Book theBook;
      while(books.hasNext()){
        theBook = (Book)books.next();
          System.out.println( theBook ); }
    } catch (StoreException e) {
        e.printStackTrace();}
  }
```

SELEZIONE DI UN OGGETTO SINGOLO

Nel caso sia chiaro che un'operazione di selezione porti ad un risultato singolo, è possibile fare uso del metodo *selectSingle()*. Anche in questo caso è possibile utilizzare una *Map* per specificare il valore delle proprietà del bean che si desidera estrarre oppure, nel caso ci sia un vincolo su un'unica proprietà, definire il tutto in un'unica invocazione che accetta proprio il nome della proprietà ed il suo valore.

```
Map filter = new HashMap();
    filter.put("year", new Integer(2005));
    filter.put("publisher", "Edizioni Master");
Book book = (Book)theStore.selectSingle(
                                   Book.class,filter);
```

Nel caso si desiderasse utilizzare un unico filtro, esiste un metodo dalla signature più "comoda" che richiede semplicemente il nome della proprietà ed il suo valore.

```
Book book = (Book)theStore.selectSingle(
          Book.class,"publisher", "Edizioni Master");
```

CONCLUSIONI

PBeans non ha certo velleità di sostituire Hibernate, ma rappresenta un'ottima soluzione, semplice e rapida per la gestione della persistenza con Java. Il suo rivale potrebbe essere Castor, ma anche in questo caso pBeans conserva alcune caratteristiche di immediatezza che lo rendono particolarmente idoneo quando non si vuole fare della persistenza il centro della propria applicazione, ma si desidera semplicemente utilizzarla al meglio.

Fabio Daniele De Michelis

POPOLARE IL DB

Molti di voi avranno la tentazione di intervenire direttamente sul DB inserendo i record relativi alle nuove istanze di libri direttamente con comandi INSERT. Quando in generale si ha a che fare con un tool di ORMapping come pBeans o altri questo è comunque un approccio sconsigliato. Prima di tutto perché se si scrive su DB mentre l'applicazione che sfrutta pBeans è in funzione, può succedere che il nuovo record non rappresenti un oggetto in pBeans il quale non ha "sentore" dell'inserimento del nuovo record. Così si può sicuramente portare il tutto in uno stato inconsistente. In linea di massima utilizzare esclusivamente Pbean per tutti gli accessi al database è la soluzione migliore.

renderci conto di come ha agito Pbean.

```
JP__OBJECTID     CLASSNAME     PERSISTENTID
------------------ ---------------- -------------------
4708896339122382358 ioprogrammo.Book
                                   6466997603929694911
2180837346892913095 ioprogrammo.Book
                                   7551407953018338209
JP__OBJECTID     PUBLISHER    TITLE              YEAR
------------------ --------- ------------------------ ----
6466997603929694911 Edizioni Master
                    Programmazione Java              2003
7551407953018338209 Edizioni Master
                 Programmare in ambiente Simbian 2005
```

OTTENERE LE ISTANZE SALVATE

In generale per eseguire un'operazione riconducibile ad una *SELECT* si utilizzano i metodi *select()* e *selectSingle()* di Store. Per ottenere tutte le istanze di una certa classe rese persistenti si invoca il metodo *select()* di store a cui si passa l'oggetto *Class,* di cui gli oggetti che desideriamo selezionare sono istanze.
Il metodo restituisce un *ResultsIterator* che racchiude tutti i bean persistenti, che può essere facilmente navigato con i relativi metodi *next()* e *hasNext()*.
È anche possibile selezionare solo quei bean che hanno un particolare valore di una proprietà. Per fare ciò è necessario costruire una *Map* che associ una proprietà di un bean al valore che deve avere, e passarla al metodo *select()* di Store; "preleverà" dal DB solo quelle istanze che soddisfano i criteri definiti dalla Map. È ovvio che molta dell'espressività possibile in una clausola *WHERE* in SQL viene persa. Il sistema di definizione dei vincoli infatti impone che il valore della proprietà debba essere proprio quello e che nel caso di due o più vincoli questi debbano essere verificati tutti contemporaneamente. Come è facile immaginare questo si traduce in una clausola *WHERE* del tipo

```
WHERE campo1 = val1 AND campo2 = val2
```

Senza possibilità di variare né l'operatore di confronto, né l'operatore logico. Di seguito il sorgente del metodo da includere nella classe *Test.*

```
public void select(){
    Map filter = new HashMap();
    filter.put("year", new Integer(2005));
    filter.put("publisher", "Edizioni Master");
    try { ResultsIterator books = theStore.select(
                                        Book.class,filter);
        Book theBook;
        while(books.hasNext()){
            theBook = (Book)books.next();
            System.out.println( theBook ); }
    } catch (StoreException e) {
        e.printStackTrace();}
}
```

SELEZIONE DI UN OGGETTO SINGOLO

Nel caso sia chiaro che un'operazione di selezione porti ad un risultato singolo, è possibile fare uso del metodo *selectSingle()*. Anche in questo caso è possibile utilizzare una *Map* per specificare il valore delle proprietà del bean che si desidera estrarre oppure, nel caso ci sia un vincolo su un'unica proprietà, definire il tutto in un'unica invocazione che accetta proprio il nome della proprietà ed il suo valore.

```
Map filter = new HashMap();
    filter.put("year", new Integer(2005));
    filter.put("publisher", "Edizioni Master");
Book book = (Book)theStore.selectSingle(
                                    Book.class,filter);
```

Nel caso si desiderasse utilizzare un unico filtro, esiste un metodo dalla signature più "comoda" che richiede semplicemente il nome della proprietà ed il suo valore.

```
Book book = (Book)theStore.selectSingle(
            Book.class,"publisher", "Edizioni Master");
```

CONCLUSIONI

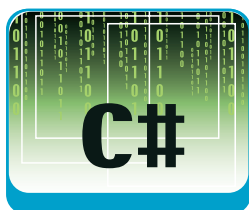
PBeans non ha certo velleità di sostituire Hibernate, ma rappresenta un'ottima soluzione, semplice e rapida per la gestione della persistenza con Java. Il suo rivale potrebbe essere Castor, ma anche in questo caso pBeans conserva alcune caratteristiche di immediatezza che lo rendono particolarmente idoneo quando non si vuole fare della persistenza il centro della propria applicazione, ma si desidera semplicemente utilizzarla al meglio.

Daniele De Michelis

POPOLARE IL DB

Molti di voi avranno la tentazione di intervenire direttamente sul DB inserendo i record relativi alle nuove istanze di libri direttamente con comandi INSERT. Quando in generale si ha a che fare con un tool di ORMapping come pBeans o altri questo è comunque un approccio sconsigliato. Prima di tutto perché se si scrive su DB mentre l'applicazione che sfrutta pBeans è in funzione, può succedere che il nuovo record non rappresenti un oggetto in pBeans il quale non ha "sentore" dell'inserimento del nuovo record. Così si può sicuramente portare il tutto in uno stato inconsistente. In linea di massima utilizzare esclusivamente Pbean per tutti gli accessi al database è la soluzione migliore.

Crystal Report Re della stampa

Primi passi nello strumento integrato da Microsoft in Visual Studio per la creazione di rapporti basati sull'aggregazione dei dati per realizzare applicazioni dedicate al business

Chi è abituato a sviluppare applicazioni legate al Business, è anche abituato ad avere a che fare con i report. Si tratta di "stampe" a video o cartaceo, che aggregano più dati e gli danno una forma statistica o ordinata tale che soddisfi le esigenze di conoscenza di un manager o di un'operatore. In sostanza i report sono spesso lo strumento ultimo per la produzione dell'output, per tale motivo assumono un'importanza assolutamente straordinaria in programmazione. Crystal Report .NET è lo strumento predisposto in Visual Studio per la creazione dei report. Supporta *ADO.NET, XML WE B Services* e *ASP.NET Server Control* ed espone un modello di programmazione semplice e ricco di funzionalità che lo rendono flessibile e funzionale. In questo articolo impareremo come sfruttare al meglio le caratteristiche di Crystal Report .NET per la creazione di report efficaci. Aprite dunque Visual Studio e andiamo a cominciare.

CREAZIONE DEL PRIMO REPORT

Creeremo un primo progetto che si chiamerà *CRlab* e lo useremo come contenitore per i nostri esperimenti:

1. aprire Visual Studio .NET e selezionare un nuovo progetto;
2. specificare *"Progetti Visual C#"* come tipi di progetto e selezionare *"Applicazione per Windows"* come modello;
3. chiamare *"CRLab"* il nuovo progetto.

Per la creazione del nostro primo report. ci serviremo dell'ormai noto database *Northwind* distribuito insieme con Microsoft SQL Server. Costruiremo un report che visualizzi in maniera tabellare i dati contenuti nel *Customer* del database Northwind. Per farlo utilizzeremo la seguente procedura:

1 fare click con il tasto destro del mouse sul nome del progetto; dal menu a tendina che appare selezionare la voce aggiungi quindi *"Aggiungi nuovo elemento";*

2 nella finestra che appare selezionare Crystal Report come tipologia di file e associarvi il nome *Customereport.rpt;*

3 premere il pulsante *Apri*;

4 nella maschera *Galleria Crystal Report* selezionare il check-box *"Utilizzo dell'esperto report"* e *"Standard"* quale tipo di report; confermare le scelte con il tasto *"ok";*

5 ora, viene visualizzata la maschera *"Esperto Report Standard",* nella cui prima scheda *"Dati"* possiamo selezionare l'origine dei nostri dati. Infatti, se si seleziona con il mouse la voce *"OLE DB (ADO)"* dalla lista delle origini dati disponibili verrà in automatico visualizzata una procedura guidata. Nella prima schermata cercare e selezionare *Microsoft OLE DB Provider for SQL Server;* nella seconda maschera selezionare il server, il database e specificare le credenziali di autenticazione. Confermare le scelte facendo click sul tasto *Avanti.* Nella ultima schermata fare click sul tasto *Fine;*

6 ritornati nella scheda iniziale, si può osservare che l'elemento *"OLE DB(ADO)"* della lista è stato popolato con un nodo corrispondente al nome del server del database. Espandendo questo nodo, si potrà notare la presenza del database *Northwind* ed, a sua volta, espandendo il quale sarà possibile fare doppio click sulla tabella *Customer* per selezionarla come sorgente dati del nostro report;

7 attivare la scheda *"Campi"* dell'*Esperto Report Standard.* In questa scheda sarà possibile scegliere i campi da visualizzare nel report. Fare click sul tasto *"Aggiungi tutto"* allo scopo di visualizzare in output tutti i campi della tabella;

8 infine, nell'ultima scheda *"Stile"* specificare il titolo *"Clienti"* e lo stile *"Standard";*

9 fare click sul tasto *"Fine"* per confermare le scelte effettuate e terminare la procedura di creazione del nuovo report.

La precedente sequenza di passi consentirà l'attivazione di Crystal Report per Visual Studio .NET. Infatti, in **Figura 1** è possibile notare la visualizzazione del Report *CustomerReport* in modalità struttura. Nella stessa figura, nella parte destra, è presente della barra *"Explorer Campo"* che consentirà la selezione di altre proprietà nel documento. In particolare, si nota la presenza della tabella *Customer* con i campi selezionati per la visualizzazione al di sotto del nodo *"Campi Database"*. In seguito alla creazione del report, nella soluzione del progetto, è stato aggiunto l'oggetto *CustomerReport* e anche la classe C# *CustomerReport*. Il fatto che venga creata una classe per ogni report, ci fa pensare che ognuno di essi possa essere instanziato a runtime, ma su questo punto si tornerà tra breve.

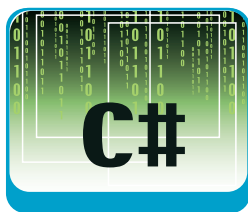
Fig. 1: **Maschera di progettazione Documento Crystal Report in Visual Studio .NET**

VISUALIZZAZIONE DI CUSTOMER REPORT

Lo schema di report creato, dovrà ora essere visualizzato nella nostra applicazione di esempio *CRLab*. Procederemo per raffinamenti successivi. Nel primo metodo di visualizzazione del report, si dovrà procedere effettuando il trascinamento di un oggetto grafico *CrystalReportViewer* nel form principale della applicazione *Form1*.
L'oggetto *CrystalReportViewer* è presente nella scheda *Windows Form* della casella degli strumenti di Visual Studio .NET.
Si proceda ora a selezionare l'elemento appena inserito e nella scheda *proprietà* modificare il valore di alcune proprietà del componente: si inizi a specificare la proprietà *name* per il nome della istanza del componente, nel nostro caso *cvwComponent;* sia settata al valore *Fill* la proprietà *Dock;*. Infine, non rimane che valorizzare la proprietà *ReportSource* all'oggetto report *CustomerReport* appena creato servendosi del controllo *browse*. Non rimane che mandare in esecuzione la applicazione *CRLab;* in tal modo il nostro primo report sarà correttamente visualizzato nella finestra principale dell'Applicazione Windows come si può osservare in **Figura 2**.
Nella stessa figura è possibile notare la barra degli strumenti nella visualizzazione. Nella barra degli strumenti sono presenti diversi controlli tra cui quello per scorrere le pagine di un report, nel caso in cui questo dovesse estendersi su più pagine; il tasto di esportazione in formato *pdf* del documento; i tasti di stampa, ricerca, e ridimensionamento della visualizzazione.

GENERARE L'ISTANZA DI UN REPORT A RUN-TIME

È stato detto in precedenza che per ogni report viene creata anche una classe; nel nostro caso è stata creata la classe *CustomerReport*. Con l'istruzione new *CustomerReport()* otteniamo una istanza della classe del report e con la quale valorizzare direttamente la proprietà *ReportSource* del componente *CrystalReportViewer* inserito nella maschera principale della applicazione *CRLab*. È necessario prima di tutto cancellare il valore inserito nella proprietà *ReportSource* del componente *cvwComponent* nella finestra di progettazione grafica. Nel metodo di gestione dell'evento *Load* del form inseriremo la se-

Fig. 2: **Visualizzazione del Report progettato**

guente linea di codice:

```
this.cvwComponent.ReportSource =
                    new CustomerReport();
```

Se si manda in esecuzione la applicazione si ottiene la visualizzazione dello stesso report come in **Figura 2**.

DATI E AUTENTICAZIONE

Durante la visualizzazione del report accade un evento un po' particolare: ogni volta viene visualizzata una maschera in cui inserire le credenziali di autenticazione per accedere ai dati della origine dati del database SQL Server. Questo accade poiché nel report sono contenute tutte le informazioni della connessione al database tranne la password. In questa sezione ci occuperemo proprio di risolvere questo fastidioso problema. Infatti, difficilmente un utente apprezzerebbe la necessità di inserire le credenziali di autenticazione al database ogni volta che viene visualizzato un report. Inoltre, una applicazione che avesse un siffatto funzionamento, non sarebbe ben progettata poiché non si potrebbe mantenere nascosta l'identità del database che contiene i dati. Per evitare tutti questi inconvenienti, è possibile aggiungere le seguenti linee di codice nel metodo di gestione dell'evento *Load* del form principale e prima della valorizzazione della proprietà *ReportSource* dell'istanza *cvwComponent:*

```
CrystalDecisions.CrystalReports.Engine.Table tbCurrent;
CrystalDecisions.Shared.TableLogOnInfo tliCurrent =
        new CrystalDecisions.Shared.TableLogOnInfo();
for(int i = 0; i < CustomerDocument.Database.Tables
                                        .Count; i++)
{ tbCurrent = CustomerDocument.Database.Tables[i];
   tliCurrent = tbCurrent.LogOnInfo;
   tliCurrent.ConnectionInfo.ServerName = serverName;
   tliCurrent.ConnectionInfo.UserID = userid
   tliCurrent.ConnectionInfo.Password = password;
   tliCurrent.ConnectionInfo.DatabaseName = "Northwind";
   tbCurrent.ApplyLogOnInfo(tliCurrent);
}
```

Questo frammento di codice consente di impostare per una istanza del report, delle credenziali di autenticazione definite a run-time. È necessario specificare i valori attuali della variabili di tipo *String* servername, userid, password che rappresentano rispettivamente il nome del computer server di database e le credenziali di accesso di un utente abilitato ad accedere al database Sql Server. Questa possibilità consente di progettare, in completa sicurezza, i report di stampa, proteggendo e nascondendo i dettagli della connessione alla sorgente dati. Se si manda in esecuzione l'applicazione si potrà notare come non venga richiesta l'immissione della password, risolvendo anche questo problema.

ISTANZE SELETTIVE

Si selezioni la scheda *Componenti* della casella degli *strumenti* e si faccia l'operazione di drag&dop del componente *ReportDocument* nell'area di visualizzazione del form principale della applicazione. In seguito a questa *operazione1,* possiamo notare che viene chiesto all'utente di specificare il tipo di report da associare al componente; a tale scopo si selezioni l'unico report presente, ovvero *CustomerReport.* Infine, agendo sulla scheda delle *proprietà* del componente, si specifichi l'oggetto *CustomerDocument*. Non ci rimane che passare alla visualizzazione del report utilizzando l'oggetto appena inserito. È sufficiente sostituire nel metodo di gestione dell'evento *Load* del form principale, l'unica istruzione con la seguente:

```
this.cvwComponent.ReportSource =
                this.CustomerDocument;.
```

Se mandiamo in esecuzione l'applicazione otteniamo lo stesso risultato dei precedenti. Questo metodo di visualizzazione di un report può essere convenientemente utilizzato allorquando ad una stessa istanza di un oggetto *CrystalReportViewer* possono essere associati, in maniera selettiva, una istanza diversa di un modello di report. Infatti, proprio in questi casi se ne apprezza l'utilità.

ALTRE MODALITÀ PER PASSARE I DATI

Analizziamo i dettagli della procedura che si deve seguire per modificare il report in maniera tale che possa prelevare i dati da un dataset, piuttosto che direttamente da un database con un *OLE DB Provider*. Il dataset sarà popolato con i dati della tabella *Customer* del database *Northwind*. Per ottenere il risultato è necessario creare un dataset vuoto:

1. sul nome del progetto *CRLab* fare click con il tasto destro del mouse;
2. selezionare aggiungi, quindi nuovo elemento;
3. scegliere *Dataset* come modello e chiamarlo *dsCustomer.xsd;*
4. premere il tasto *Apri* per terminare la procedura.

Per popolare il dataset con i dati della tabella *Customer,* è necessario fare doppio click sul nodo *dsCustomer.xsd* nella scheda *Esplora soluzioni* per aprire una nuova finestra di progettazione; selezionare la scheda *Esplora Server,* espandere il nodo corrispondente al nome del proprio computer, espandere il nodo *SQL Server*, selezionare e trascinare la tabella *Customer* del database *Northwind* all'interno della finestra di progettazione di *dsCustomer.xsd* e salvare il file. A questo punto aprire il file del report *CustomerReport,* fare click con il tasto destro e selezionare *Database* e quindi *"Imposta posizione…".* Nella maschera *"Imposta Posizione"* che appare, selezionare dalla lista a destra il nodo *"Dati di Progetto";* espandere *Datasets ADO.NET,* quindi selezionare il dataset

Fig. 3: ***Modifica della origine dati al report***

e infine la tabella *Customers*. A questo punto espandere il nodo *"Connessioni correnti"* selezionare *dsCustomer* quindi fare click sul tasto *"sostituisci"*.
Ora che abbiamo modificato la sorgente dati del nostro report, è necessario creare una istanza del dataset *dsCustomer* e assegnarlo alla proprietà *ReportSource* dell'oggetto visualizzatore nel form principale. Per raggiungere questo obiettivo eseguire i passi qui riportati:

1. da *Esplora* server selezionare trascinare la tabella *Customer* nella finestra di design del form principale. Questa operazione aggiungerà gli oggetti *SQLConnection* e *SQLDataAdapter* al form;
2. l'oggetto *SQLConnection* venga rinominato *Northwindconn*, mentre l'oggetto *SQLDataAdapter* come sdaCustomers;
3. l'oggetto di connessione al database *Northwindconn*, venga modificato dalla scheda della *proprietà* per aggiungere il parametro *pwd*, ovvero una password valida per l'autenticazione nel database;
4. fare click con il tasto destro del mouse sull'oggetto *sdaCustomer* e dal menu selezionare la voce *"Genera Dataset.."*. Comparirà la maschera *"Genera Datase..t"*, dalla quale selezionare il checkbox corrispondente ad un dataset esistente (il nostro dataset *dsCustomer*), quindi fare click su *OK* per confermare la scelta;
5. nella finestra di design del form, sarà inserita la istanza del dataset corrispondente alla tabella *Customer* (ossia *dsCustomer1*);
6. per associare al visualizzatore del report il dataset appena creato occorrerà scrivere e sostituire al metodo di gestione dell'evento *Load* del form le seguenti linee di codice:

```
sdaCustomers.Fill(dsCustomer1);
CustomerDocument.SetDataSource(dsCustomer1);
this.cvwComponent.ReportSource =
                              CustomerDocument;
```

7. nandare in esecuzione l'applicazione Windows. Nel form viene visualizzato correttamente il *Report*.

METODO DELLA STORED PROCEDURE

In questa sezione ci proponiamo di assegnare all'oggetto *ReportDocument* una istanza di un oggetto dataset ottenuto per mezzo della seguente stored procedure parametrizzata.

```
Create PROCEDURE dbo.spCustomers
(@CustPattern nVarChar(40))
AS
select * from Customers Where CompanyName Like
                                 @CustPattern + '%'
RETURN
```

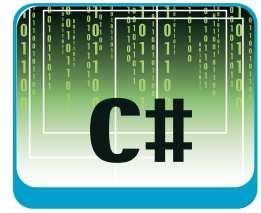
A questo punto è sufficiente sostituire le seguenti linee di codice al metodo di gestione dell'evento *Load* del form:

```
System.Data.SqlClient.SqlCommand
                          cmdCustomersSP = new
                System.Data.SqlClient.SqlCommand(
                   "spCustomers",this.NorwindConn);
cmdCustomersSP.CommandType =
                   CommandType.StoredProcedure;
cmdCustomersSP.Parameters.Add("@CustPattern", "A");
System.Data.SqlClient.SqlDataAdapter sdaCustomersSP
          = new System.Data.SqlClient.SqlDataAdapter(
                                  cmdCustomersSP);
DataSet dsReport = new DataSet();
sdaCustomersSP.Fill(dsReport, "Customers");
CustomerDocument.SetDataSource(dsReport);
cvwComponent.ReportSource = CustomerDocument;
```

Dal codice possiamo notare che alla stored procedure è stato assegnato il valore *"A"* per filtrare i dati della tabella *Customer*.
L'aspetto interessante da notare dal frammento di codice precedente, è il fatto che è stato utilizzato il solo oggetto connessione *NorwindConn*. In si può notare la visualizzazione attuale del report con i dati filtrati in base al parametro fissato a tempo di compilazione.

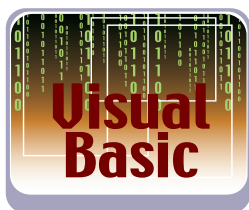
CONCLUSIONI

Crysal Reports è uno strumento particolarmente potente e complesso. In questo articolo abbiamo mostrato solo la punta dell'iceberg fornendovi le basi per iniziare a utilizzare questa importante funzionalità.
Tuttavia avete già a disposizione tutte le conoscenze per creare dei report anche complessi utilizzando semplicemente le caratteristiche intrinseche di SQL.
Datevi da fare e stupite i vostri clienti con report che mettono a nudo il loro business.

Elmiro Tavolaro

NOTA

Il codice allegato alla rivista *CrystalReportLab.zip* va scompattato in una directory del computer di test. Il file rappresenta il progetto realizzato in Visual Studio .NET e illustrato passo-passo durante il presente articolo. Affinché il tutto funzioni è necessario effettuare la creazione del report secondo i passi illustrati nel corso dell'articolo.

Media Player con Visual Basic

L'SDK per estendere Windows Media Player contiene alcuni controlli da utilizzare all'interno del nostro ambiente preferito per realizzare un software per la gestione del multimedia

Tutti conosciamo Windows Media Player. Il software incluso in Windows è utilizzato per visualizzare contenuti multimediali come filmati e suoni. Pochi sanno che esiste un *Microsoft Windows Media Player Software Development Kit (MWMP SDK)*, che serve ad estenderne le capacità. La versione più recente dell'SDK è la 10. In questo articolo utilizzeremo l'*Activex Windows Media Player*, contenuto nella libreria *WMP.dll* per creare un lettore di DVD e CD-Rom. Data la vastità dell'argomento nei nostri esempi utilizzeremo soltanto le proprietà *URL* e *Close* sufficienti per i nostri scopi. In particolare la *URL*, serve per specificare il nome di un file contenuto in una periferica locale o remota mentre *Close* serve a rilasciare il controllo. WMP per accedere ai DVD e ai CD-Rom si poggia sui protocolli *WmpDvd* e *WmpCD*. I comandi elaborati da questi protocolli devono essere scritti in sintassi *URL*. Per esempio per accedere ad un DVD è necessario usare la seguente forma:

```
WindowsMediaPlayer1.URL =
        wmpdvd://drive/title/chapter?contentdir=path
```

Dove *Drive*, è la lettera che identifica il lettore DVD (senza i classici due punti), *Title* è il numero del titolo che si vuole eseguire, *Chapter* è il numero del capitolo, *contentdir=path* è il path di un file *VIDEO_TS.IFO*, file che contiene informazioni sul contenuto del DVD.

LETTORE MP3 E REGISTRATORE WAV

Il form del lettore Mp3 è presentato in **Figura 1**. Il form, con dei frame, è diviso in due parti: *player* e *registratore*. Per il player sono previsti i seguenti elementi: quattro pulsanti - *Open*, *Play*, *Stop* e *Pausa* - un textbox che contiene il titolo del brano riprodotto, una label che fornisce informazioni sul tempo di esecuzione e uno oggetto *Slider* per controllare la posizione corrente del lettura. Per il registratore, invece, sono previsti: tre pulsanti - *Record*, *Stop* e *Salva* - e una label che fornisce informazioni sullo stato della registrazione. Inoltre, sul form, sono presenti due pulsanti (+ e -) per controllare il livello del volume degli altoparlanti, un pulsante per ricercare i file da riprodurre, due oggetti *Timer* utilizzati per controllare lo stato dell'esecuzione *(TimerPlayer)* e della registrazione *(TimerRecord)* e un *CommonDialog* necessario per la fase di ricerca e salvataggio dei file.

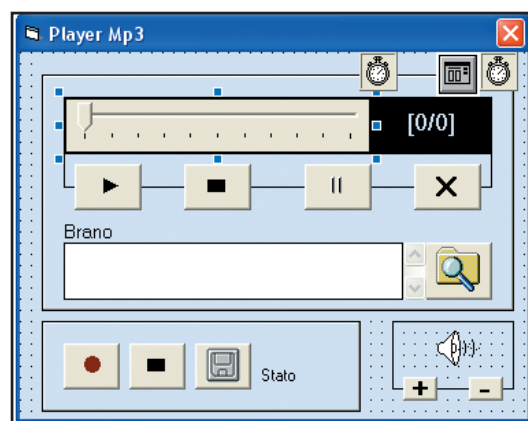



Fig. 1: ***Il nuovo PlayerMp3 in fase di progettazione***

IL REGISTRATORE WAV

Tramite la funzione *MciSendString*, di cui abbiamo parlato ampiamente nel numero 93 di ioProgrammo, la registrazione di un nuovo file *Wav* si avvia con una sequenza, di tre comandi base: *Open New*, per aprire il device; *Set*, per impostare i parametri di registrazione e *Record* per l'avvio vero e proprio della registrazione. Si

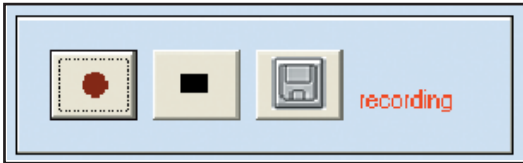



Fig. 2: Il registratore in fase recording

fa notare che un comando *Set* deve essere seguito dal nome del parametro che si vuole impostare, per esempio con *Set Format* per impostare il formato dell'ora, *Set channels* per il Numero di canali, *Set bitspersample* per il numero di Bit del campionamento, *Set samplespersec* per la Frequenza di campionamento ecc. Queste istruzioni sono contenute nella procedura del pulsante *Record* mostrata di seguito.

```
Private Sub Record_Click()
 Dim tipo As String
 Dim i As Integer
 Frameplayer.Enabled = False
 tipo = "Waveaudio"
 mciSendString "open new type " & tipo & " alias "
                                    & mioalias, 0, 0, 0
 i = mciSendString("set mioalias time format
                                milliseconds", 0, 0, 0)
 mciSendString "set " & mioalias & " samplespersec
                                     44000", 0, 0, 0
 mciSendString "set " & mioalias & " channels 2",
                                            0, 0, 0
 mciSendString "set " & mioalias & " bitspersample
                                        16", 0, 0, 0
 mciSendString "record " & mioalias, 0, 0, 0
 Timerrecord.Interval = 1
End Sub
```

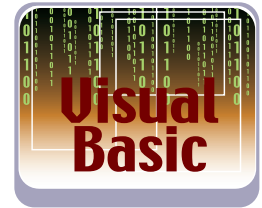
Notate che nella *Record_Click* si disattiva il frame del *Player* e si attiva il *TimerRecord*, il quale esegue le istruzioni che su una label (nominata *Labelrecord*) mostrano lo stato della registrazione, recuperato con comando base *Status*, come mostrato in seguito.

```
Private Sub Timerrecord_Timer()
 Dim i As Long, Buf As String, s As Single
 Buf = Space$(128)
 i = mciSendString("status " & mioalias _
                          & " mode ", Buf, 128, 0)
 If Mid(Buf, 1, 3) = "rec" Then
  Labelrecord.ForeColor = &HFF&
 Else
  Labelrecord.ForeColor = &H80000012
 End If
  Labelrecord = Buf
End Sub
```

Per terminare la registrazione si usa il comando base *Stop*

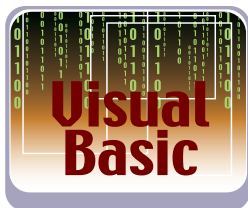
```
Private Sub StopRecord_Click()
 mciSendString "stop " & mioalias, 0, 0, 0
 Frameplayer.Enabled = True
 Timerrecord.Interval = 0
End Sub
```

Nella *StopRecord_Click* dopo l'invio del comando *Stop* viene abilitato il *FramePlayer*, per ascoltare il file registrato prima di salvarlo.
Il salvataggio è fatto con la *Salva_Click* che, tra l'altro, invoca la procedura *salvafile* che esegue il comando *MCI Save* e chiude la periferica aperta.

```
Private Sub Salva_Click()
 Dim nomefilecorto As String
 eseguirecord = False
 With CommonDialog1
  .Filter = "file Wav |*.wav"
  .ShowSave
  Dim pos As Integer
  pos = InStrRev(.filename, "\")
  Dim com As String
  com = Mid(.filename, 1, pos)
  SalvaFile com
  Timerrecord.Interval = 0
 End With
End Sub
Private Sub SalvaFile(nome As String)
 Dim nomefilecorto As String
 nomefilecorto = pathnomecorto(nome)
 mciSendString "save " & mioalias & " " & _
 nomefilecorto + CommonDialog1.FileTitle, 0, 0, 0
 mciSendString "close " & mioalias, 0, 0, 0
 Frameplayer.Enabled = True
End Sub
```

Utilizza questo spazio per le tue annotazioni

RICAVARE IL PATH

La funzione pathnomecorto che ci consente di recuperare il nome abbreviato del file e la dichiarazione della mciSendString indispensabile per inviare i vari comandi.

```
Attribute VB_Name = "Module1"
Public Declare Function mciSendString
   Lib "winmm.dll" Alias "mciSendStringA"
 (ByVal lpstrCommand As String, ByVal
      lpstrReturnString As String, ByVal
             uReturnLength As Long, ByVal
            hwndCallback As Long) As Long
Public Declare Function
 mciGetErrorString Lib "winmm.dll" Alias
 "mciGetErrorStringA" (ByVal dwError As
 Long, ByVal lpstrBuffer As String, ByVal
                uLength As Long) As Long
Public Declare Function
     mciSendCommand Lib "winmm.dll"
      Alias "mciSendCommandA" (ByVal
   wDeviceID As Long, ByVal uMessage
     As Long, ByVal dwParam1 As Long,
      ByVal dwParam2 As Long) As Long
Public Declare Function
   GetShortPathName Lib "kernel32" _
Alias "GetShortPathNameA" (ByVal
               lpszLongPath As String, _
ByVal lpszShortPath As String, ByVal
           cchBuffer As Long) As Long

  Public Function pathnomecorto(ByVal
   sLongFileName As String) As String
  Dim lRetVal As Long, sShortPathName
              As String, iLen As Integer
  sShortPathName = Space(1200)
  iLen = Len(sShortPathName)

  lRetVal = GetShortPathName(
      sLongFileName, sShortPathName,
                                  iLen)
  pathnomecorto = Left(
             sShortPathName, lRetVal)
  End Function
```

La *SalvaFile* utilizza la funzione *pathnomecorto*, introdotta nel precedente appuntamento, per ricavare il nome corto del file da salvare.

UN LETTORE DI VIDEOCLIP

Il lettore di video Clip presenta una grafica spartana e i soli comandi base - *Cerca file, Apri,* e *Stop* - che abilitano le omonime funzioni. Per realizzare questa parte, come sempre, nel progetto, è necessario referenziare soltanto la libreria che contiene l'oggetto *CommandDialog*. Per quanto riguarda il Form su di esso dovete prevedere: un frame, che servirà come area di visualizzazione del video; tre pulsanti nominati *Cerca, Play* e *Stop;* un textbox che contiene il nome del file riprodotto e naturalmente un *CommandDialog*.

Fig. 3: **Il Player di VideoClip**

Nella parte dichiarativa del form inseriamo la dichiarazione della funzione *mciSendString* (che trovate nel CD) e quella delle seguenti variabili.

```
Dim Com As Long, Buf As String
```

La variabile *Com* conterrà il valore restituito dalla *MciSendString*. La *Buf*, invece, è il Buffer utilizzato per interagire con il device.
La procedura relativa al pulsante *Cerca* è semplice e non la presentiamo, quella relativa al pulsante *Play* invece è la seguente.

Fig. 4: **Il Form del Player di VideoClip**

```
Private Sub Play_Click()
 Stop_Click
 Buf = Space$(128)
 com = mciSendString("Open " & txtnomefile & "
                                    TYPE MPEGvideo" _
 & " ALIAS video parent " & Frame1.hWnd & " style
                                    child", Buf, 128, 0)
 If com Then MsgBox "File non riconosciuto",
                                    vbCritical, "Errore"
 com = mciSendString("Play video", Buf, 128, 0)
End Sub
```

Nella *Play_Click* notate che al comando base Open sono state aggiunte le parole chiave *Parent* e *style child* che servono rispettivamente ad indicare il controllo collegato al comando (il *Frame1*) e lo stile di visualizzazione. I comandi *Play* e *Stop*, invece, non cambiano (restano come quelli del PlayerMp3).

```
Private Sub Stop_Click()
 Dim i As Long, Buf As String
 Buf = Space$(128)
 i = mciSendString("stop video", Buf, 128, 0)
 i = mciSendString("close video", Buf, 128, 0)
End Sub
```

UN LETTORE DI DVD E CDROM

Il lettore presentato in questo paragrafo funziona soltanto con il sistema operativo Windows XP, dato che si base sul controllo Windows Media Player versione 10.

Fig. 4: **La form del progetto apparirà come in figura**

Per realizzare l'esempio create un nuovo progetto e referenziate la libreria *WMP.dll*. Sul *Fom1* inserite il controllo *Win-dowsMediaPlayer1* e un menu a discesa con i seguenti comandi di primo livello: *Apri File, CDAudio, DVD* ed

NOTE

TIPI E PROTOCOLLI SUPPORTATI DA WMP

Windows Media Player, attualmente alla versione 10, supporta vari protocolli e molti tipi di file. Questi possono essere utilizzati con l'Activex WMP impostando opportunamente la proprietà URL. I principali formati supportati sono: *ASF (Advanced Systems Format), AIF, MPE, MPEG, MP3, WAV, (Windows Media Audio), WMV (Windows Media Video).* Tra i protocolli supportati ricordiamo: *HTTP, RTSP (Real Time Streaming Protocol), WMPCD* e *WMPDVD.* Questi ultimi sono utilizzati per accedere direttamente ai CDRom e ai DVD.

Eject. Sulla finestra dell'editor di menu dovete impostare i valori per i TextBox *Caption* e *Name* rispettivamente con il nome del menu (*Apri-File, CDAudio* ecc.) e il nome della relativa procedura (*mnuapri, mnucdaudio,* ecc). Le procedure dei tre menu sono le seguenti.

```
Private Sub mnuapri_Click()
 On Error Resume Next
 With Me.CommonDialog1
   .ShowOpen
   If .filename <> "" Then
     WindowsMediaPlayer1.close
     WindowsMediaPlayer1.URL = .FileTitle
   End If
 End With
End Sub
Private Sub mnucdaudio_Click()
 WindowsMediaPlayer1.URL = "wmpCD://0/1"
End Sub
```

Notate che la *mnucdaudio* per aprire i CD utilizza il protocollo *wmpCD* e che *0* (zero), in questo caso, sostituisce la lettera *D:* (che identifica l'unità CD)

```
Private Sub mnuDVD_Click()
 Me.WindowsMediaPlayer1.URL = "wmpdvd://D/1/1"
End Sub
```

Nella *mnuDVD_Click* invece è utilizzato il protocollo *wmpdvd.* Notare che *D* identifica l'unità DVD e che con *"1/1"* identifica il primo titolo e il primo capitolo.

APRIRE IL DVD E DIMENSIONARE IL FORM

Per aprire il supporto del lettore utilizziamo la collezione *cdromCollection* in cui sono presenti gli indirizzi dei Driver dei CdRom e dei DVD del nostro computer.

```
Private Sub mnueject_Click()
 WindowsMediaPlayer1.cdromCollection. _
 getByDriveSpecifier(D).eject
End Sub
```

Anche in questo caso la lettera *D* identifica il lettore CD o DVD. Per ridimensionare il Form e il controllo WMP utilizziamo il seguente codice.

```
Private Sub Form_Resize()
 On Error Resume Next
 WindowsMediaPlayer1.Left = 0
 WindowsMediaPlayer1.Top = 0
 WindowsMediaPlayer1.Height = Me.Height - 800
 WindowsMediaPlayer1.Width = Me.Width - 100
End Sub
```

Nella *Form_Resize* notate che i valori 800 e 100 servono per togliere le dimensioni del bordo del controllo WMP (questi naturalmente possono essere valutati a *Run_Time,* come abbiamo fatto per l'esempio della WebCam).

CONTROLLARE UNA WEBCAM

In questo paragrafo illustriamo come visualizzare le immagini riprese da una WebCam e come catturarle e salvarle in un file.

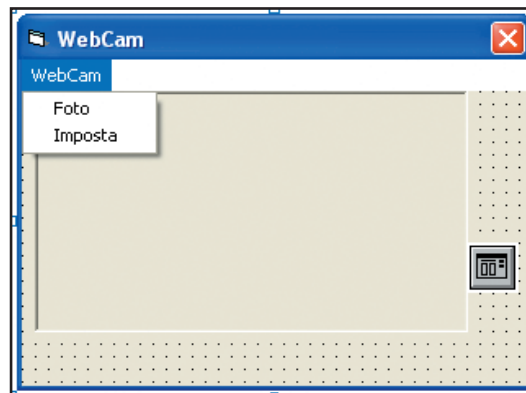



Fig. 5: ***Ecco come appare la form che utilizzeremo per impostare e catturare le immagini***

Sul form inseriremo un PictureBox, un controllo *CommandDialog* e un menu a discesa, quest'ultimo deve avere i comandi *Foto* e *Imposta* che consentono rispettivamente di catturare l'immagine della WebCam, visualizzata sul PictureBox, e di abilitare la finestra *impostazioni WebCam.* La *Form_Load* si presenta come segue:

```
Dim hWndCam As Long
Dim Hi As Integer
```

WAVEOUTGETVOLUME E LA WAVEOUTSETVOLUME

Le funzioni API utilizzate per controllare il volume del lettore Mp3 sono: *waveOutGetVolume* e *waveOutSetVolume.* La prima restituisce il livello del volume, la seconda lo imposta, entrambe hanno due parametri: l'handle del device audio aperto e un puntatore a una Word di 32 bit che contiene le impostazioni del volume. In particolare, la parte bassa di questa Word, contiene le impostazioni del volume per il canale sinistro, mentre la parte alta contiene le impostazioni per il canale destro. Il livello massimo del volume è dato dal valore *&HFFFF&,* quello più basso da *&H0000&.* Attenzione! Non tutti i device supportano queste funzioni o i due canali.

MASTERIZZARE UN CD-ROM

Per masterizzare un CD con Visual Basic si possono utilizzare due tecniche, quella proprietaria basata sulle Image Mastering API (IMAPI) e quella basata su *NeroSDK-v1.05.* Le IMAPI, fornite con Windows XP, permettono di masterizzare CD-R e CD_RW. Le IMAPI in Visual Basic 6 sono utilizzabili solo con l'ausilio di uno Wrapper C++ (involucro - che definisce le interfacce cioè le IDL per VB). *Nero Burning ROM,* il più diffusi software di masterizzazione, può essere inserito nelle nostre applicazioni grazie a *NeroSDK-v1.05* che mette a disposizione dei programmatori le *DLL: NeroCOM* e *NeroAPI.* Quest'ultima è utilizzabile solo dai programmatori C++, la *NeroCOM* invece può essere utilizzato anche con Visual Basic 6, dato che si tratta di un Wrapper nato intorno a *NeroAPI.dll. v1.05* è scaricabile gratuitamente da www.nero.com ma può essere utilizzato solo se sul computer è installata la versione completa di *Nero Burning ROM.*

```
Dim Wi As Integer
Dim DifHi As Integer
```

La variabile *hWndCam* conterrà l'handle dell'area *cattura immagini*, in *Hi* e *Wi;* invece, verrà inserita la risoluzione corrente della WebCam, contenuta nel tipo *CapStatus*. *DifHi* sarà impostata con la differenza di *Height* tra il form e l'area attiva al suo interno, questo valore verrà utilizzato nella *Resize*.

```
Private Sub Form_Load()
 WebCam.Align = 1
  'aligntop
 WebCam.AutoRedraw = True
 WebCam.AutoSize = True
 DifHi = Me.Height - Me.ScaleHeight
 WebCam.ScaleMode = 3
 inizializza
End Sub
```

MCISENDSTRING

Con la funzione *MciSendString* negli esempi, sono inviati, anche, i comandi base: Set e Status. Il comando *Set* imposta alcuni parametri dei device multimediali, quali formato dei canali video e audio, formato del tempo, canali sonori ecc. Il comando Status, invece, serve per recuperare informazioni sui *Device*, questo comando a differenza del *Set* è supportato da tutti i device. La seguente istruzione, per esempio, permette di conoscere la lunghezza del brano musicale che si sta riproducendo.

```
mciSendString("status " & mioalias & "
                        length", Ris, 128, 0)
```

Il dato richiesto è restituito nella variabile Ris, 128 è la lunghezza di *Ris*.

IMPOSTARE L'AREA DI CATTURA

La procedura *Inizzializza*, invocata dalla *Form_Load*, definisce l'area di cattura, con la *capCreateCaptureWindow*, la connette alla WebCam ed imposta e attiva il modo *Preview*.

```
Private Sub Inizializza()
 hWndCam = capCreateCaptureWindow(
  "WebCamCat", _WS_CHILD Or WS_VISIBLE, 0, 0, _
 WebCam.Width, WebCam.Height, WebCam.hWnd, 0)
 'crea una finestra figlia (CHILD) visibile
 If hWndCam <> 0 Then
  SendMessage hWndCam,
                        WM_CAP_DRIVER_CONNECT, 0, 0
  SendMessage hWndCam,
                        WM_CAP_SET_PREVIEWRATE, 60, 0
  SendMessage hWndCam, WM_CAP_SET_PREVIEW,
                                    CLng(True), 0
 End If
 DimensioneCattura
End Sub
```

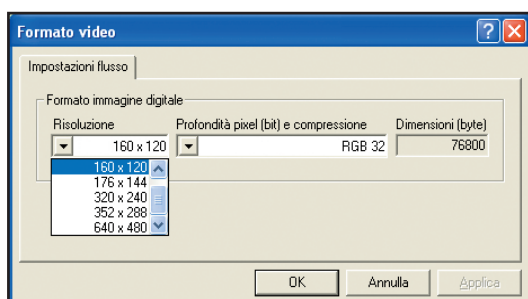



Fig. 6: La form contenente i controlli per l'impostazione dei parametri

La procedura Inizializza, a sua volta, invoca la *DimensioneCattura* per impostare le variabili *Hi* e *Wi* con le dimensioni dell'immagine catturata dalla WebCam.

```
Private Sub dimensionecattura()
 SendMessage hWndCam, WM_CAP_GET_STATUS, _
 Len(WebCamstate), WebCamstate
 Wi = WebCamstate.uiImageWidth
 Hi = WebCamstate.uiImageHeight
 Caption = CStr(Wi) + "  " + CStr(Hi)
End Sub
```

IMPOSTARE LA WEBCAM E SCATTARE FOTO

Le procedure invocate dal menu a discesa sono la *MnuImposta* e la *MnuFoto*.

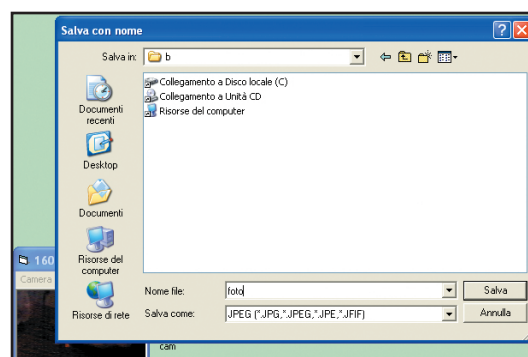



Fig. 7: Subito dopo avere scattato salveremo la foto sull'Hard Disk

```
Private Sub MnuImposta_Click()
 SendMessage hWndCam,
                WM_CAP_DLG_VIDEOFORMAT, 0&, 0&
 DimensioneCattura
 Form_Resize
End Sub
```

Notare che la *MnuImposta* oltre ad avviare la finestra delle impostazioni della WebCam invoca le procedure che rilevano ed impostano

le dimensioni dell'area di cattura e del form, cioè la *DimensioneCattura* e la *Form_Resize*

```
Private Sub mnufoto_Click()
 Dim filename As String
 SendMessage hWndCam, WM_CAP_SET_PREVIEW, _
                                    CLng(False), 0&
 With dlgSalvataggio
  .Filter = "JPEG (*.JPG) |*.JPG|"
  .ShowSave
  filename = .filename
 End With
 SendMessage hWndCam, WM_CAP_FILE_SAVEDIB, _
                           0&, ByVal CStr(filename)
 SendMessage hWndCam, WM_CAP_SET_PREVIEW, _
                                     CLng(True), 0&
End Sub
```

RIDIMENSIONAMENTO E CHIUSURA DEL FORM

Come accennato, nei passi precedenti, il ridimensionamento del Form è fatto in base alla risoluzione della WebCam contenute nelle variabili *Hi* e *Wi*.
Facciamo notare che *Hi* e *Wi* sono impostate in Pixel mentre le dimensioni del form sono in *Twip* e che la conversione da *Pixel* in *Twip* viene fatta moltiplicando per il valore 15.

```
Private Sub Form_Resize()
 Me.Height = Hi * 15 + DifHi
 Me.Width = Wi * 15
 WebCam.Height = Hi * 15
 Me.Refresh
End Sub
Private Sub Form_Unload(Cancel As Integer)
 On Error Resume Next
 SendMessage hWndCam,
                    WM_CAP_DRIVER_DISCONNECT, 0, 0
End Sub
```

Nella *Form_Unload* viene disconnessa la WebCam.

CONCLUSIONI

È stato un appuntamento ricco di esempi che sicuramente hanno ampliato le vostre conoscenze sul fantastico mondo della multimedialità. Questi spunti andrebbero approfonditi! Magari cercando di capire come si possono archiviare i filmati prodotti da una WebCam e come si può utilizzare al meglio il Controllo Windows Media Player.

Massimo Autiero

LE API PER GESTIRE LA WEBCAM

Descriviamo le funzioni API, le costanti e i tipi da utilizzare per catturare i frame di una WebCam, questi elementi appartengono alle librerie *avicap32.dll* e *user32.dll* e permettono di controllare una finestra "cattura immagini" - *CaptureWindow* - attraverso dei semplici messaggi. In particolare, gli elementi che utilizziamo sono la *SendMessage,* la *capCreateCaptureWindow,* alcuni messaggi ed il tipo *CAPSTATUS,* quest'ultimo contiene informazioni sullo stato della WebCam. Le dichiarazioni dettagliate di questi elementi si trovano nel modulo del progetto allegato alla rivista.
La *SendMessage* è una funzione utilizzata in vari contesti e permette di controllare il comportamento di una o più window. I parametri della funzione sono i seguenti: *hWnd,* l'identificatore della finestra che riceve il messaggio; *wMsg,* il messaggio da spedire; *Wparam* e *Lparam* il primo e il secondo argomento del messaggio. Negli esempi sono utilizzati i messaggi descritti nella *Tabella 1*.
Facciamo notare che con il messaggio *WM_CAP_GET_STATUS* si recupera lo stato della finestra di cattura che è contenuto nella struttura dati *CAPSTATUS.* Questa contiene diversi dati tra i quali *uiImageWidth* (ampiezza della cattura) e *uiImageHeight* (altezza della cattura) che utilizzeremo per conoscere la risoluzione corrente della WebCam.
La *capCreateCaptureWindow,* invece, permette di creare la superficie di cattura e di associarla ad un oggetto Visual Basic. La *capCreateCaptureWindow* ha i seguenti parametri: *lpszWindowName* per specificare il nome della superficie o finestra di cattura; *dwstyle* per specificare lo stile della finestra; *X,Y* per impostare la posizione; *nWidth* e *nHeigth* per le dimensioni; *hwndParent* per indicare la finestra che contiene i frame catturati ed infine *nID* che serve per specificare l'identificatore della finestra di cattura.
La *capCreateCaptureWindow* come risultato restituisce l'handle della finestra di cattura. Come vedremo, nel nostro esempio, la finestra di cattura è associata ad una *PictureBox* nominata *WebCam.*
La tabella seguente riassume alcuni dei messaggi per comandare una WebCam

WM_CAP_GET_STATUS	Restituisce lo stato della finestra di cattura nella struttura CAPSTATUS.
WM_CAP_DRIVER_CONNECT	Connette la finestra di cattura al drive che gestisce la WebCam.
WM_CAP_DRIVER_DISCONNECT	Disconnette la finestra di cattura dal drive.
WM_CAP_SET_PREVIEW	Abilita oppure disabilita il modo preview. In altre parole permette il trasferimento dei frame dalla WebCam alla memoria e quindi alla finestra di visualizzazione.
WM_CAP_SET_PREVIEWRATE	Imposta, in millisecondi, la quantità di frame da trasferire nel modo preview.
WM_CAP_DLG_VIDEOFORMAT	Mostra un dialogbox con il quale l'utente può selezionare il formato video.
WM_CAP_FILE_SAVEDIB	Copia il frame corrente in un file DIB, cioè un file BMP che contiene informazioni sulle sue dimensioni.

Java gestisce il customer care

Vi presentiamo un modello per la creazione di un helpdesk automatico. Legge la richiesta di assistenza, crea un ticket, lo assegna ad un operatore e vi regala un cliente soddisfatto!

L'ipotetica azienda ABC Informatica desidera creare un sistema di gestione del customer care semiautomatizzata. Fornisce ai suoi clienti un indirizzo di posta, a cui inviare tutte le segnalazioni, richieste di intervento ed altre informazioni tecniche. Questo indirizzo di posta, *abcinformatica@nomedominio.ext* viene controllato ad intervalli regolari da una applicazione Java che, tramite JavaMail, scarica tutti i messaggi in arrivo. Ciascun messaggio viene inserito in un database di richieste ed il mittente viene notificato da un sistema automatico che gli comunica dell'avvenuta ricezione del messaggio, e del numero di ticket assegnato alla richiesta. All'interno di ABC Informatica il customer care ha a disposizione un'applicazione Web che gli consente di tenere sotto controllo i ticket in arrivo, quelli in lavorazione ed eventualmente anche quelli terminati.

CREARE LA BASE DATI

Il database utilizzato sarà *mySQL* mentre per la connessione da java utilizzeremo JDBC. La tabella che conterrà i campi avrà la struttura illustrata in **Tabella 1**.

Campo	Tipo	Chiave
ID_TICKET	INT(11)	PK
MITTENTE	VARCHAR(30)	JDX1
DATA_RICEZIONE	DATE	IDX2
OGGETTO	VARCHAR(255)	
TESTO	LONGTEXT	
DATA_LETTURA	DATA	
UTENTE_INCARICO	VARCHAR(10)	IDX3
STATO	INT	
DATA_CHIUSURA	DATA	

Tabella 1:

Gli stati possibili per ciascun ticket saranno solo tre:

1. **aperto.** Questo stato viene assegnato ai messaggi in ingresso, al momento della loro ricezione.
2. **in lavorazione.** Un ticket viene inserito in questo stato quando è stato letto la prima volta da un utente, che diviene automaticamente l'utente in carico. Praticamente, quando qualcuno legge un ticket aperto per la prima volta, gli viene automaticamente assegnato.
3. **chiuso.** Il ticket viene chiuso semplicemente quando l'utente che lo ha in carico decide che questo non richiede altro lavoro. Tutto quello che accade tra lo *stato 2* ed il *3* è a discrezione dell'operatore di customer care. Ovviamente, in un sistema reale, questo non sarebbe sufficiente, ma in questo caso basta per provare l'architettura del progetto.

Per ogni stato viene memorizzata la data/ora in cui il ticket entra in quello stato. Viene memorizzata la data di creazione, di lettura e di chiusura del ticket.

LEGGERE E MEMORIZZARE LE RICHIESTE

Il cuore dell'applicazione è la classe *MailReader*, che si occupa di scandagliare un account di posta ad intervalli regolari e di leggere i messaggi in arrivo. Questi vengono poi inseriti in un database, e viene inviato un messaggio di conferma al mittente. La classe definisce una serie di attributi che definiscono l'url di connessione al database, il mittente per il messaggio di risposta, l'host del server pop3, l'account di posta da controllare e la relativa password:

```
public class MailReader {
    protected final int REFRESH = 5000;
    protected String databaseUrl =
                "jdbc:mysql://127.0.0.1/assistenza";
    protected String accountNoReply =
                        "noreply@nomedominio.ext";
    protected String hostName = "mail.nomedominio.ext";
    protected String account = "max@ nomedominio.ext ";
    protected String password = "*****";
```

Inoltre, sono presenti un oggetto *Folder* che rappresenta la casella di posta in ingresso, una sessione di posta rappresentata da un oggetto *Session* ed una connessione al database, sotto forma di oggetto *Connection:*

```
    protected Folder inboxFolder;
    protected Session mailSession;
    protected Connection databaseConnection;
```

il costruttore della classe è il luogo dove è implementato il ciclo di lettura. Il metodo *connect()* stabilisce la connessione al database, *getInboxFolder()* ritorna la casella di posta in ingresso mentre *readMessages()* legge ed elabora i messaggi:

```
    public MailReader() throws InterruptedException,
                                        MessagingException,
        IOException, ClassNotFoundException,
                                        SQLException {
            connect();
            inboxFolder = getInboxFolder();
            while (true) {
                readMessages();
                Thread.sleep(REFRESH);
            }
    }
```

la connessione al database avviene caricando in memoria il driver tramite la chiamata a *Class.forName()* ed utilizzando *DriverManager* per ottenere una connessione. Essendo questo un programma standalone non viene utilizzato *DataSource*, come invece sarà per l'applicazione Web di consultazione dei ticket:

```
    protected void connect() throws
                ClassNotFoundException, SQLException {
        Class.forName("com.mysql.jdbc.Driver");
        databaseConnection = DriverManager.getConnection(
                        databaseUrl, "root", "mysql" );
    }
```

la cartella dei messaggi in arrivo viene invece ottenuta utilizzando le API *JavaMail*. È necessario per prima cosa ottenere un oggetto *Session* valido, ottenuto con il metodo *getDefaultInstance()*. Questo si aspetta un oggetto *Properties* che contiene eventuali elementi di configurazione della sessione. In questo caso viene impostato l'host smtp di output utilizzato per l'invio dei messaggi. Una volta in possesso della sessione è possibile ottenere un oggetto *Store*. In questo caso si chiama *"pop3"*, in quanto il server chiamato è di questo tipo. Il metodo *connect()* permette di eseguire la reale connessione al server. Il cui nome di host, insieme ad utente e password, vengono passati come parametri. Una volta connesso lo Store è possibile ottenere la cartella di default e da questa la cartella *INBOX*. Questo è il nome predefinito quando si desidera accedere ad un server pop3. In caso di server imap è necessario invece passare il nome della cartella da leggere.

```
protected Folder getInboxFolder() throws
                                MessagingException {
    Properties props = System.getProperties();
    props.put( "mail.smtp.host", "mail.nomedominio.ext" );
    mailSession = Session.getDefaultInstance(props, null);
    Store store = mailSession.getStore("pop3");
    store.connect( hostName, account, password );
    Folder folder = store.getDefaultFolder();
    if (folder != null) {
        folder = folder.getFolder("INBOX");
    }
    return folder;
}
```

il metodo *readMessages()* si occupa di leggere e gestire i messaggi. La prima operazione è l'apertura della cartella dei messaggi in arrivo con il metodo *open()*. La cartella è apribile in sola lettura o in lettura/scrittura. Con il metodo *getMessages()* si ottiene invece l'elenco dei messaggi in arrivo. A questo punto è sufficiente scorrere il vettore ed elaborare ciascun messaggio che viene passato ad *insertMessage()* per l'inserimento nella base dati ed a *sendResponse()* per l'invio della risposta. Per comodità viene estratto il mittente del messaggio con il metodo *getFrom()*. Questo viene poi passato ai suddetti metodi:

```
protected void readMessages() throws SQLException,
  MessagingException, IOException {
    inboxFolder.open(Folder.READ_WRITE);
    Message[] elencoMessaggi =
                        inboxFolder.getMessages();
    System.out.println("leggo i messaggi...");
    for (int indice = 0; indice
                < elencoMessaggi.length; indice++) {
      Message messaggio = elencoMessaggi[ indice ];
      InternetAddress fromAddress =
```

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

JAVAMAIL

Le API JavaMail permettono, all'interno della piattaforma Java, di inviare e ricevere messaggi di posta utilizzando i protocolli POP3, IMAP e SMTP. I messaggi possono essere di tipo MIME e composti da più parti, come nel caso di messaggi con allegati.

```
                (InternetAddress)messaggio.getFrom()[0];
        String from = fromAddress.getPersonal();
        if( from == null ) {
            from = fromAddress.toString();
        }
        System.out.println("inserisco messaggio da " +
                                                    from);
        String ticketID = insertMessage( messaggio,
                                                   from );
        System.out.println("invio risposta a " + from);
        sendResponse( messaggio, ticketID );
        // messaggio.setFlag(Flags.Flag.DELETED, true);
        // messaggio.saveChanges();
        // System.out.println("cancello messaggio");
    }
    System.out.println("fine.");
    inboxFolder.close(false);
}
```

al termine delle operazioni il folder viene chiuso. Si noti che per cancellare un messaggio correttamente elaborato è possibile impostare il flag *DELETED* a *true* e salvare le modifiche. Questa operazione non è però disponibile con server pop3 ma solo con imap. Per questo motivo il codice è presente ma commentato.

INSERIMENTO DEI MESSAGGI E RISPOSTA

Il metodo *insertMessage()* si aspetta un messaggio ed una stringa con il mittente. L'inserimento avviene tramite API JDBC che eseguono uno statement preparato che esegue una *INSERT* sulla tabella *TICKET*. Il testo del messaggio viene estratto con il metodo *getTesto()*, in quanto questa è una operazione un po' complicata.

```
protected String insertMessage( Message messaggio,
                                          String from )
    throws SQLException, MessagingException,
                                          IOException {
        String ticketID = null;
        String sql = "INSERT INTO TICKET " +
        " (MITTENTE, DATA_RICEZIONE, OGGETTO,
                                          TESTO, STATO) "
        + " VALUES (?,SYSDATE(),?,?,1) ";
        PreparedStatement insertStatement =
            databaseConnection.prepareStatement(sql);
        insertStatement.setString(1, from);
        insertStatement.setString(2,
                              messaggio.getSubject());
        insertStatement.setString(3, getTesto(messaggio));
        insertStatement.execute();
        insertStatement.close();
        sql = "SELECT LAST_INSERT_ID() AS T_ID";
        Statement queryStatement =
            databaseConnection.createStatement();
        ResultSet rs = queryStatement.executeQuery(sql);
        if (rs != null) {
            if (rs.next()) {
                ticketID = rs.getString(1);
            }
        }
    return ticketID;
}
```

al termine dell'inserimento viene utilizzata la funzione specifica di MySQL per ottenere l'ultimo ID inserito, in quanto questo è proprio un campo definito come auto_increment nel database. L'ID del ticket viene ritornato dal metodo. Il metodo *getTesto()* controlla che il tipo sia text /plain e se è così imposta un ciclo di lettura su uno stream ottenuto dall'oggetto *Messaggio*. Questo ritorna il contenuto che, riga per riga, viene memorizzato in un buffer:

```
protected String getTesto( Message messaggio )
    throws MessagingException, IOException {
        StringBuffer body = new StringBuffer();
        if( messaggio.getContentType()
                            .startsWith("text/plain") ) {
            InputStream in = messaggio.getInputStream();
            BufferedReader reader = new BufferedReader(
                                new InputStreamReader( in )
            );
            do {
                String linea = reader.readLine();
                if( linea == null ) {
                    break;
                }
                body.append( linea );
            } while( true );
            reader.close();
            in.close();
        } else {
            body.append("impossibile leggere il testo
                                          della mail");
    }
    return body.toString();
}
```

le API *JavaMail* richiedono di leggere il testo di un messaggio in questo modo per offrire un meccanismo generico che permetta di leggere anche dati binari, come gli allegati. Un messaggio può essere infatti anche di tipo "multipart", essere quindi costituito da testo normale, testo html, immagini ed allegati. Questa versione del programma supporta dunque solo mail in ingresso di tipo assolutamente te-

NOTA

UN COMPLETO FRAMEWORK

JavaMail è anche un framework generico per la posta. Non è ristretto all'utilizzo dei protocolli presenti in Internet. Può anche essere esteso per gestire altri protocolli, come Microsoft Exchange o Lotus Notes. Si potrebbero supportare anche altri tipi di messaggi, che non rispettano lo standard MIME.

stuale. Il metodo che invia la risposta utilizza ancora le API *JavaMail*, ma questa volta per l'invio. È possibile creare un messaggio di posta da inviare istanziando un oggetto *MimeMessage*, a cui viene passato un oggetto *Session*. Questo oggetto ha metodi per impostare il soggetto, il testo ed altri elementi, come il mittente. Quest'ultimo è rappresentato da un oggetto *InternetAddress*, che si costruisce semplicemente passando l'indirizzo di posta, come ad esempio *noreply@nomedominio.ext*.

```
protected void sendResponse( Message messaggio,
                                        String ticketID )
  throws MessagingException {
    String text = "* questo è un messaggio
                            automatico *\r\n\r\n" +
    "La sua richiesta di intervento è stata registrata " +
    "con ticket " + ticketID;
    Message message = new MimeMessage(
                                      mailSession );
   InternetAddress from = new InternetAddress(
                                   accountNoReply );
   InternetAddress to = (InternetAddress)
                              messaggio.getFrom()[0];
   message.setFrom( from );
   message.addRecipient(
                    Message.RecipientType.TO, to );
   message.setSubject( "Conferma di ricezione" );
   message.setSentDate( new java.util.Date() );
   message.setText( text );
   try {
     Transport.send(message);
   } catch(SendFailedException ex) {
       System.out.println("impossibile inviare il
                   messaggio ("+ex.toString() + ")");
  }
}
```

l'invio avviene tramite la chiamata a *Transport.send()*.

PRESENTARE LE INFORMAZIONI

Una volta che i dati sono acquisiti e memorizzati nel database è possibile fare qualsiasi operazione, ad esempio visualizzazioni, gestioni, statistiche.
Un'operazione indispensabile è quella di visualizzazione dei ticket aperti (quelli con *stato 1*). Un modo semplice per ottenere questa funzionalità è tramite una pagina JSP come la seguente:

```
<%@ page language="java" %>
<%@ page import="java.util.*" %>
<%@ page import="net.ioprogrammo.assistenza.*" %>
[Omissis]
<jsp:useBean id="ticketBean" scope="session" class=
        "net.ioprogrammo.assistenza.TicketBean" />
<h1>Visualizzazione ticket ricevuti</h1>
<table width="100%">
<%
  List tickets = ticketBean.getElencoTicketAperti();
  for( Iterator iter = tickets.iterator();
  iter.hasNext(); ) {
    TicketInfo currentTicket = (TicketInfo)iter.next();
    out.write("<tr>");
    out.write("<td>" + currentTicket.getId() +
                                         "</td>");
    out.write("<td><a href=\"\">" +
        currentTicket.getMittente() + "</a></td>");
    out.write("<td>" + currentTicket.getOggetto() +
                                         "</td>");
    out.write("</tr>");
  }
  %>
</table>
```

Il *JavaBean ticketBean* fornisce il metodo *getElencoTicketAperti()* che ritorna una lista di oggetti *TicketInfo*, ciascuno dei quali contiene un diverso ticket caricato dal database. Scorrendo l'elenco è possibile creare una tabella dei ticket aperti.
L'oggetto *TicketInfo* è un semplice contenitore di informazioni definito come segue:

```
package net.ioprogrammo.assistenza;
import java.util.*;
public class TicketInfo {
  public static final int APERTO = 1;
  public static final int IN_LAVORAZIONE = 2;
  public static final int CHIUSO = 3;
  long id;
  String mittente;
  Date dataRicezione;
  String oggetto;
  String testo;
  Date dataLettura;
  String utenteInCarico;
  int stato;
  Date dataChiusura;
  public TicketInfo(long id, String mittente, Date
     dataRicezione, String oggetto, String testo, Date
   dataLettura, String utenteInCarico, int stato, Date
                                   dataChiusura) {
    super();
    this.id = id;
    this.mittente = mittente;
    this.dataRicezione = dataRicezione;
    this.oggetto = oggetto;
    this.testo = testo;
    this.dataLettura = dataLettura;
```

NOTA

TUTTO IN UN MESSAGGIO

La classe *Message* definisce un messaggio di posta generico e supporta un numero illimitato di mittenti e destinatari. Tramite opportuni metodi permette di conoscere informazioni sul messaggio, come la data di ricezione, di invio, il numero progressivo.

```
        this.utenteInCarico = utenteInCarico;
        this.stato = stato;
        this.dataChiusura = dataChiusura;
    }
    public Date getDataChiusura() {
        return dataChiusura;
    }
    public void setDataChiusura(Date dataChiusura) {
        this.dataChiusura = dataChiusura; }
    //… omissis, nel codice sono presenti getter e
    //setter perciascuna proprietà
}
```

RICERCHE E FILTRI

In JavaMail è presente anche un completo sistema di ricerca e selezione dei messaggi nelle cartelle di posta. Queste funzionalità sono presenti nel package *javax.mail.search.* Questo include classi per la ricerca su mittenti, destinatari, date, dimensioni ed altri elementi del messaggio. La classe *Folder* dispone del metodo *search(),* che ritorna un array di messaggi che corrispondono ad un determinato termine di ricerca.

la lettura di questi oggetti è affidata alla classe *TicketBean*. Questa classe dispone del metodo *locateDataSourcePool()* che si occupa di ottenere la connessione al database secondo lo standard J2EE. Il nome del datasource è contenuto nella costante *DATASOURCE_NAME* e deve essere definito nell'application server in uso. In questo caso specifico il programma è stato provato su JBoss 3.2.4 e nel relativo file *mysql-ds.xml* è stato definito il data source in questo modo:

```
<local-tx-datasource>
    <jndi-name>jdbc/assistenza</jndi-name>
    <connection-url>jdbc:mysql://127.0.0.1:3306
                            /assistenza</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>root</user-name>
    <password>****</password>
  </local-tx-datasource>
```

il codice per ottenere il data source da codice è il seguente:

```
void locateDataSourcePool() throws NamingException {
    if( dataSource == null ) {
        Context initContext = new InitialContext();
        Context envContext =
                    (Context)initContext.lookup("java:");
        Object o = envContext.lookup(
                                DATASOURCE_NAME );
        dataSource = (javax.sql.DataSource)o;
        if( dataSource == null ) {
            throw new NamingException("Il datasource " +
              DATASOURCE_NAME + " non è stato definito" );
            }
        }
}
```

a questo punto è possibile leggere i ticket con una semplice query:

```
public List getElencoTicketAperti()
    throws SQLException, NamingException {
        locateDataSourcePool();
        Connection databaseConnection =
                        dataSource.getConnection();
        List result = new ArrayList();
        String sql = " SELECT " + " ID_TICKET,
            MITTENTE, DATA_RICEZIONE, OGGETTO, " +
          " TESTO, DATA_LETTURA, UTENTE_INCARICO,
              STATO, " +" DATA_CHIUSURA " +" FROM "
             + " TICKET " + " WHERE " + "   STATO=1";
        Statement queryStatement =
                databaseConnection.createStatement();
        ResultSet rs = queryStatement.executeQuery(sql);
        if (rs != null) {
            while (rs.next()) {
                result.add( createTicket(rs) );}
    }
    return result;
}
```

mentre il metodo di supporto *createTicket()* accorpa la chiamata necessaria a creare un oggetto *TicketInfo*. Questo metodo viene richiamato per ogni ticket letto dalla base dati:

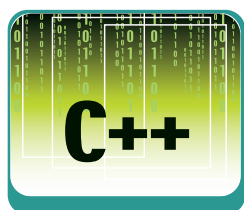
```
TicketInfo createTicket( ResultSet rs ) throws
                                    SQLException {
    int pos = 1;
    return new TicketInfo( rs.getLong(pos++),
            rs.getString(pos++),
            rs.getDate(pos++),
            rs.getString(pos++),
            rs.getString(pos++),
            rs.getDate(pos++),
            rs.getString(pos++),
            rs.getInt(pos++),
            rs.getDate(pos++));
    }
}
```

CONCLUSIONI

Abbiamo illustrato lo scheletro di un programma di helpdesk, certamente può essere esteso, ma si tratta di un'applicazione con ottimo valore commerciale da tenere in considerazione quando qualcuno dei vostri clienti vi chiederà di implementare qualcosa.

Massimiliano Bigatti

Compilatori Intel veloci come la luce

Intel Integrated Performance Primitives. Come sfruttare ogni più piccola possibilità offerta dal processore per generare applicazioni ultraveloci. Un viaggio nell'extreme programming

Nei numeri precedenti di ioProgrammo, abbiamo già affrontato le caratteristiche peculiari dei compilatori Intel riguardo a produzione di codice talmente ottimizzato da risultare incredibilmente veloce rispetto a quello prodotto dai compilatori standard. In particolare abbiamo visto come si possa generare codice a diversi livelli di prestazioni, a seconda di quanta attenzione si vuol dare alle direttive di compilazione e di quanto si vuole mirare ad un'architettura specifica. L'analisi dell'output assembler generato dal compilatore ha dimostrato come, anche all'interno della famiglia Intel, esistono grandi differenze di approcci disponibili, a seconda che si possano utilizzare o meno diversi registri o particolari instruction set (IA-32, MMX, o i vari SSE). Tanta meticolosa attenzione alle prestazioni dà certamente i suoi frutti, ma il punto fondamentale è che è assolutamente inutile se non ne si ha bisogno. Ottimizzare è positivo, infatti, solo quando si ha un'effettiva necessità di prestazioni elevate. La manipolazione dei segnali digitali, delle immagini e il *"number crunching"* sono esempi classici di campi che condividono questo bisogno, anche perché sono quelli in cui è possibile avvalersi maggiormente di vettorizzazione e parallelismo, sfruttando tecnologie avanzate come l'*hyper threading*, o le istruzioni *SIMD*. Si pensi, ad esempio, al caso in cui si abbia un'immagine, e ne si voglia aumentare la luminosità. Questo consiste nell'effettuare un incremento del valore d'ogni pixel, nel modo seguente:

```
for (char *pixel = primopixel; pixel != ultimopixel;
                                            pixel++)
        *pixel += incremento;
```

Fig. 1: Il programma in esecuzione

Questo procedimento farà probabilmente suonare un campanello d'allarme in molti di voi: è, infatti, il tipico caso in cui è possibile effettuare una *vectorization*, rendendo il più possibile parallelo (e quindi più rapido), un processo che per sua natura lavorerebbe serialmente su ogni singolo pixel.

Quando si lavora nel campo dell'audio e delle immagini, si ha a che fare costantemente con situazioni di questo tipo, giacché si opera per la maggior parte del tempo su vettori e matrici. *Intel Integrated Performance Primitives* (d'ora in poi IPP) è una libreria molto vasta, che ha lo scopo di fornire supporto a questo genere di problemi.

CARATTERISTICHE DI IPP

IPP è una libreria cross-platform, disponibile sia per Linux sia per Windows, che integra una serie di funzionalità nei campi più svariati; in generale, i domini di riferimento sono quattro:

- **Ipps:** gestione dei vettori (*1D*), per i quali sono fornite funzionalità d'inizializzazione, logiche, aritmetiche e di filtraggio. Inoltre esistono fun-

zioni specifiche per la manipolazione delle stringhe, dell'audio, dei segnali, per la codifica del parlato e il riconoscimento vocale.

- **Ippi:** gestione di immagini (*2D*), per le quali sono fornite funzionalità lineari, statistiche, geometriche, morfologiche, logiche, aritmetiche, di thresholding e filtraggio. Copre anche le codifiche JPEG, le trasformazioni Wavelet, la computer vision e molti protocolli di codifica video (i vari formati MPEG, H.263, H.264, etc…).
- **Ippm:** gestione di piccole matrici, per le quali sono fornite funzionalità particolarmente ottimizzate per il calcolo vettoriale e matriciale, soluzione di sistemi lineari, e molte altre.
- **Ippcp:** dominio dedicato alla crittografia a chiave simmetrica (*DES* e *triplo DES, Rijndael, Blowfish, Twofish*), alle funzioni di *hash* (*MD5* e i vari *SHA*) e a chiave pubblica (funzioni per i numeri primi, pseudocasuali, etc…).

In quest'articolo ci occuperemo innanzitutto di installare e integrare correttamente la libreria con Visual Studio, impareremo i fondamenti della programmazione in IPP, e, per finire, analizzeremo qualche funzione del dominio dedicato alle immagini.

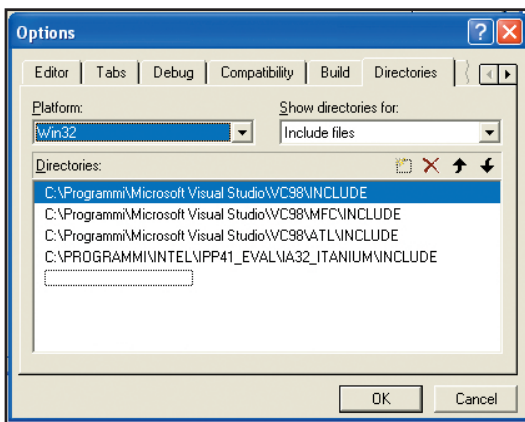



Fig. 2: Il pannello di controllo per settare le opzioni di compilazione

DOWNLOAD ED INSTALLAZIONE

Analogamente al compilatore, le IPP sono disponibili sia per Linux (per il quale sono gratuite, per progetti non commerciali) sia per Windows (al costo di 200$). Qui ci occuperemo di installare le librerie sotto Windows, nella loro versione di valutazione gratuita per 30 giorni, e di integrarle con Visual Studio 6. Nonostante ciò, bisogna considerare che la differenza in termini di codice fra le due versioni è praticamente inesistente, e tutto ciò che vedremo (a parte, ovviamente, i riferimenti a GDI), si applicherà senza variazioni anche per la versione Linux. L'installazione della libreria sarà un probabile déjà-vu per chi ha seguito attivamente il nostro ultimo appuntamento: dobbiamo, infatti, scaricare la versione dimostrativa della libreria (107 Mb), all'indirizzo *http://www.intel.com/software/products/ipp/downloads/ippwin_eval.htm* e installare il prodotto, previa registrazione alla pagina *https://registrationcenter.intel.com/EvalCenter/EvalForm.aspx?ProductID=263*. L'installazione in sé non presenta alcun aspetto particolare, se si esclude la proposta (da accettare) di configurazione automatica delle variabili d'ambiente. Al termine, troveremo tutto quanto ci serve nella cartella *"C:/Programmi/Intel /IPP41_eval/"*. D'ora in avanti chiamerò *IPPROOT* la relativa sottocartella *"…/Ia32_Itanium"*. Per integrare le librerie in Visual Studio dobbiamo raggiungere la linguetta di selezione dei percorsi di *Tools /Options…* ed inserire nella lista degli include il percorso *"IPPROOT /Include"*. Discorso analogo per i percorsi delle librerie, ai quali aggiungeremo *"IPPROOT /Stublib"*. Siamo quasi pronti per cominciare a scrivere la nostra applicazione d'esempio: sceglieremo un nuovo progetto MFC basato su finestre di dialogo, cui - con molta fantasia! - daremo il nome di *IPPEsempio* (questo, quantomeno, è quel che ho fatto io, nel file che trovate allegato alla rivista).

DISPATCHING DINAMICO

Prima di cominciare a scrivere istruzioni, dobbiamo scegliere un modello di linking. Come vedremo ben presto, le *IPP* sono state progettate per ottenere il massimo delle performance, anche a costo di aumentare la ridondanza del codice. Per capire la realtà della situazione, provate ad esplorare la cartella *IPPROOT/Bin:* troverete una serie di *DLL* il cui nome segue il modello *"Ipp[Dominio]20.dll"* – come, ad esempio, *Ippi20.dll, Ippcore20.dll*, etc…
Queste prendono il nome di *"Dispatcher"*, e devono sempre trovarsi in un percorso del path. Il loro compito è smistare le chiamate alle librerie specializzate. Se analizzate, infatti, il contenuto della sottocartella *"Ipp20"*, troverete molte altre librerie (ben più pesanti, in termini di spazio occupato), il cui nome segue il pattern *"Ipp[dominio][sigla processo-*

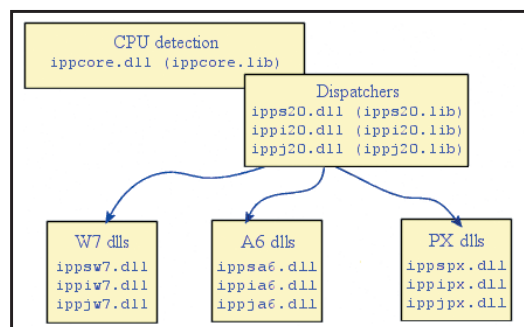
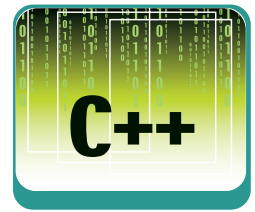
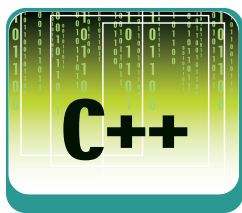



Fig. 3: Schema del modello di linking dinamico

Utilizza questo spazio per le tue annotazioni

COMPUTER VISION

Le funzionalità per la computer vision di IPP sono veramente potenti e performanti. La libreria open source OpenCV, una delle più apprezzate del settore, si basa proprio sulle ippi per i suoi calcoli.

re]", sulla stregua di *Ippia6.dll, Ippii7.dll,* etc… Ognuna di queste contiene il vero codice delle *IPP:* tutte le varie funzioni che compongono la libreria, infatti, sono state progettate con algoritmi differenziati per ogni diverso modello di processore. La **Figura 3** mostra quello che avviene in caso di dispatching dinamico (il modello che sceglieremo noi). A runtime, la libreria *Ippcore.dll* si occupa di determinare il modello del processore della macchina su cui il programma viene eseguito. Quindi i vari dispatchers smistano le chiamate alla libreria più appropriata. Questo modello è il più semplice ed efficace, ed anche il più pesante in termini di deploy di un'eventuale applicazione (è necessario, infatti, ridistribuire tutte le sottolibrerie). Per metterlo in atto bisogna comunicare al linker di appoggiarsi sulle librerie *stub* presenti nella directory che abbiamo già incluso in precedenza. Tutto ciò si riduce, quindi, ad andare in *Project/Settings…,* nella linguetta *"link",* e scrivere il nome delle librerie interessate (nel nostro caso solo la *ippi20.lib*) in *object modules*

NOTA

MANUALE

Difficilmente ci si potrebbe spingere molto al largo in quell'oceano di funzioni che costituisce le IPP, senza i manuali relativi a ogni dominio. Questi sono molto completi e disponibili nella cartella *"…/Ipp41_eval/Doc/".*

STRUTTURE E FUNZIONI IN IPP

Chi inizia a muovere i primi passi nel mondo delle *IPP* si trova solitamente un po' disorientato; è una sensazione che passa presto: una volta capita la filosofia di fondo su cui si appoggia questa libreria, tutto sembra giusto e naturale. Il primo elemento da comprendere è la denominazione dei tipi di dati: Ipp ridefinisce i tipi di dato primitivi (come char, int, etc…) secondo una sintassi standard: *Ipp[numeroDiBit][u|s|f][c],* a seconda che il tipo sia senza segno *(u),* con segno *(sgned),* o in virgola mobile *(f).* Il suffisso *c,* inoltre, indica un numero complesso. Così, ad esempio, *Ipp8u* corrisponde a un byte senza segno, *Ipp16s* a un intero *16-bit* con segno *Ipp32f* ad un 32-bit in virgola mobile, e *Ipp32sc* ad un numero complesso con parti intere a 32-bit con segno. Come avrete già immaginato dalla moltiplicazione delle librerie per ogni singolo processore, l'attenzione che viene dedicata alle performance è quasi maniacale: la logica che governa il sistema è quella "del rasoio": *IPP* fornisce funzioni atomiche, che svolgono compiti singoli, indivisibili, il più possibile specifici. È quindi classico che una funzione (prendiamo *ippiCopy,* ad esempio) si ritrovi ad avere una ventina di prototipi distinti: esistono, infatti, *ippiCopy_8u_, ippiCopy_16s_, ippiCopy_32f_,* a seconda che l'operazione si svolga su dati *8u, 16s,* o *32f.* Si dice in gergo che le varie funzioni appartenenti alla famiglia *Copy* sono 'decorate', ognuna in modo diverso. Una funzione viene decorata dal tipo di dato cui si applica, ma non solo. In *ippi* la maggior parte dei prototipi segue questo schema:

```
ippiFunzione_TipoDiDato_[C|P][numeroCanali/Piani][I
][R][Altrelettere]
```

Un esempio è *ippiMirror_8u_C1R,* che indica la funzione di ribaltamento di un'immagine di tipo *8u,* con un solo canale *(C1)* e possibilità di definire una regione d'interesse (*R* sta per *Region of Interest,* o *ROI*). La funzione *ippiMirror_16s_C3IR* è analoga alla precedente, con la differenza del tipo di dato, del fatto che i canali stavolta sono tre, e che l'operazione avviene sull'immagine stessa, e non più su una di destinazione (*I,* infatti, sta per *In Place*). Quando, al posto di *C,* si trova la lettera *P,* questo vuol dire che l'immagine non è organizzata su canali, bensì su piani distinti.

GESTIONE DELLA MEMORIA

L'ultimo aspetto da comprendere a fondo se si vogliono evitare errori grossolani, è la rappresentazione in memoria di un'immagine allocata con *IPP.* Un'immagine può essere memorizzata su canali o su piani, e può possederne uno (monocromatica), tre (a colori) o quattro (a colori con alpha). Le sezioni *b* e *c* della **Figura 5** mostrano la differenza fra

Fig. 5: ***Disposizione dei byte di un'immagine (a) in memoria, secondo la rappresentazione a canali (b), a piani (c), e a canali con allineamento a 32 bit (d).***

memorizzazione a canali e a piani.
Un punto cruciale riguarda l'allineamento dei byte: dal momento che le istruzioni *SIMD* lavorano più velocemente quando la memoria è allineata a 32 bit, ad ogni riga che compone l'immagine vengono aggiunti tanti byte quanti ne servono perché si ottenga un multiplo di 32. Questi byte inutili (chiamati in gergo *padding bytes*) non sono inizializzati, ed introducono una disparità fra la larghezza dell'immagine e l'effettivo numero di bytes per ogni riga (chiamato solitamente *stepBytes*), che va sempre tenuta in considerazione. In **Figura 5d** è possibile osservare la

reale disposizione che assumerebbero in *IPP* i byte dell'immagine d'esempio. Nonostante nello stesso manuale si indulga talvolta (per ragioni di spazio) su tale punto, un'immagine dovrebbe sempre essere allocata richiamando la funzione *ippiMalloc,* che allinea automaticamente la memoria in modo corretto. Nel nostro esempio ci limiteremo a lavorare solo con immagini *8u* a tre canali, che corrispondono alle classiche bitmap a 24 bit, quindi una figura di 100x200 pixel verrà allocata nel seguente modo:

```
int stepBytes;
Ipp8u *img = ippiMalloc_8u_C3(100, 200,
                                        &stepBytes);
```

Img sarà così il puntatore al primo elemento dell'immagine, la cui riga consisterà di 300 byte dati (uno per ogni canale). A questi, saranno aggiunti automaticamente 20 byte di padding, per fare in modo che il totale della riga (320) sia multiplo di 32. IppiMalloc è molto gentile, e ci evita di fare il conto (coi nostri mezzi rudimentali e lenti), restituendoci automaticamente nel terzo parametro il numero di bytes per riga. In questo caso, dunque, *stepBytes* sarà automaticamente inizializzato a 320. Se avete capito tutto questo, è giunto il momento di premiare la vostra resistenza a tanta teoria: passiamo alla pratica!

INIZIALIZZAZIONE DEL PROGETTO

Il nostro progetto di esempio dovrà servire a familiarizzare con la libreria, perciò non sarà tanto complicato. Si limiterà solo a caricare una bitmap 24 bit da file, ed eseguirvi sopra delle trasformazioni geometriche, qualche filtro, e qualche analisi statistica. Cominciamo dalla finestra di dialogo: poniamo una casella di testo *(IDC_NOMEFILE)* per l'ingresso del percorso del file da aprire, un frame *(IDC_IMMAGINE)* per la visualizzazione su schermo dell'immagine, ed un pulsante *(IDC_CARICA)* per il caricamento dell'immagine da file. Il punto cruciale (in cui, peraltro, le strade Windows e Linux si allontanano inesorabilmente) sta nel caricamento dell'immagine e la relativa visualizzazione su schermo. Tutto questo richiede una certa conoscenza delle GDI, che do per scontata. Ciononostante, cercherò lo stesso di illustrare i passaggi potenzialmente più oscuri. Nel file *CIPPEsempioDlg.h,* aggiungiamo:

```
#include <ippi.h>
```

e, all'interno della classe:

```
Ipp8u*   img; // Puntatore all'immagine IPP
int  imgStep;  // Numero di bytes in una riga di img
Ipp8u*   bmp; // Puntatore all'immagine BMP
int bmpStep;  // Numero di bytes in una riga di bmp
IppiSize  size; // Dimensioni dell'immagine
CDC  imgDC; // Contesto di dispositivo che contiene
                                                 bmp
void ImgToBmp(); // Passa il contenuto di Img in Bmp
void BmpToImg(); // Passa il contenuto di Bmp in Img
```

Queste righe dovrebbero essere chiare. Esisteranno due immagini, in memoria: *img* sarà usata da IPP, e *bmp* da GDI. Non possiamo usare, infatti, lo stesso buffer, per una disparità di allineamento: IPP allinea a 32 byte, mentre GDI a 4. Dovremo quindi prevedere delle routine di interscambio mediante le quali passare i dati da *img* a *bmp (ImgToBmp)* e viceversa *(BmpToImg).* Questa è la loro implementazione, che ritengo essere autoesplicativa:

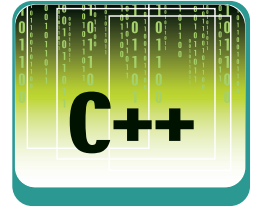
```
void CIPPEsempioDlg::ImgToBmp() {
  for (int y=0; y<size.height; y++)
  memcpy(bmp + (y*bmpStep), img + (y*imgStep),
                                    size.width*3);
}
void CIPPEsempioDlg::BmpToImg() {
  for (int y=0; y<size.height; y++) memcpy(img
              + (y*imgStep), bmp + (y*bmpStep),
                                    size.width*3);
}
```

In questo tipo di approccio dobbiamo tener presente una cosa: nelle bitmap 24 bit Windows le terne non sono rappresentate in RGB, bensì in BGR (ovvero: prima il byte del blu, poi del verde, poi del rosso), e le righe sono rovesciate (ovvero da quella più in basso a quella più in alto). Tuttavia, dal momento che IPP non si aspetta di avere necessariamente un rosso sul primo canale, e poiché la nostra rappresentazione viene "raddrizzata" direttamente da GDI, questo non costituisce un problema.
Per finire, aggiungiamo l'inizializzazione delle variabili in *OnInitDialog:*

```
img = bmp = NULL;
GetDlgItem(IDC_NOMEFILE)->
          SetWindowText("C:/Percorso/Prova.bmp");
```

CARICAMENTO DELL'IMMAGINE

Per restare sul semplice, faremo in modo che possano essere caricati solo file BITMAP a 24 bit. In questo modo possiamo avvalerci della comoda funzione *LoadImage:*

```
void CIPPEsempioDlg::OnCarica() {
  //Carico il file
  CString FilePath;
```

SUL WEB

ESEMPI

Alcuni esempi sono forniti assieme al pacchetto d'installazione, ed altri possono essere scaricati all'indirizzo http://www.intel.com/software/products/ipp/samples.htm. Sono da considerarsi molto di più che semplici esercizi: sono estremamente dettagliati, e implementano spesso soluzioni molto sofisticate da copiare e incollare, o da adattare alle proprie esigenze.

```
GetDlgItem(IDC_NOMEFILE)->
                    GetWindowText(FilePath));
HBITMAP hBitmap = (HBITMAP)LoadImage(
            NULL, FilePath, IMAGE_BITMAP, 0, 0,
    LR_LOADFROMFILE | LR_CREATEDIBSECTION |
                            LR_DEFAULTSIZE);
//Ne ottengo le informazioni
BITMAP bm;
GetObject (hBitmap, sizeof (BITMAP), &bm);
(bm.bmBitsPixel != 24) {
  MessageBox("Questo programma tratta solo
                        bitmap a 24 bit");
  return;
}
size.width = bm.bmWidth;
size.height = bm.bmHeight;
bmp = (Ipp8u *)bm.bmBits;
bmpStep = bm.bmWidthBytes;
```

Giunti a questo punto dobbiamo fare in modo che *ImgDC* contenga la *bmp:* su questo punto conto sulla vostra conoscenza di GDI.

```
//Creo il CDC di scambio
ImgDC.DeleteDC();
ImgDC.CreateCompatibleDC(GetDlgItem(
                    IDC_IMMAGINE)->GetDC());
CBitmap CBmp;
CBmp.DeleteObject();
CBmp.Attach(hBitmap);
ImgDC.SelectObject(&CBmp);
```

Ora si tratta di caricare in memoria *img* e copiarvi il contenuto della bitmap.

```
  //Carico img in memoria
  if (img) ippiFree(img);
  img = ippiMalloc_8u_C3(size.width,
                    size.height, &imgStep);
  Copy();
  //Visualizzo i dati
  Blit();
}
```

in quest'ultima parte potete notare la chiamata a *ippiMalloc,* e l'antagonista *ippiFree,* per il rilascio della memoria allocata.
La funzione *Blit(),* infine, effettua un *bitblitting* del contenuto sul frame di destinazione, ed è definita così:

```
void CIPPEsempioDlg::Blit() {
        CDC *TempDC = GetDlgItem(
                    IDC_IMMAGINE)->GetDC();
        TempDC->BitBlt(0,0,size.width,size.height,
                    &ImgDC,0,0,SRCCOPY);
        TempDC->DeleteDC();
}
```

FUNZIONI ARITMETICHE E GEOMETRICHE

A questo punto la nostra applicazione è in grado di visualizzare una bitmap 24 bit. Ora cominceremo ad arricchirla di funzionalità interessanti. Creiamo un altro pulsante con testo: *"capovolgi in verticale",* con il quale rovesceremo l'immagine, lungo l'asse delle *x*. Questo è il codice dell'handler:

```
void CIPPEsempioDlg::OnMirrorY()  {
        ippiMirror_8u_C3IR(img, imgStep, size,
                        ippAxsHorizontal);
        ImgToBmp();
        Blit();
}
```

Tutto qui. La riga interessante, ovviamente è la prima istruzione: secondo il "canone classico" il nome della funzione implica il tipo di dato, di immagine e l'operazione *'In place':* il prototipo è questo:

```
ippiMirror_<mod>(const Ipp<datatype>* pSrcDst,
      int srcDstStep, IppiSize roiSize, IppiAxis flip);
```

Per quanto possano apparirvi strani, provate a familiarizzare con questo genere di prototipi: nel nostro caso *<mod>* è uguale a *8u_C3IR*, e *<datatype>* è *8u. DstStep* sono gli *stepBytes, roiSize* è la zona d'interesse (tutta), e *IppiAxis* un'enumerazione con *ippAxsHorizontal, ippAxsVertical* o *ippAxsBoth.* Se avete capito questa funzione… complimenti: non avrete più problemi ad interpretare le centinaia (sì: sono tante) di altre funzioni presenti nel manuale delle *ippi!* Proviamo, invece, ad aumentare la luminosità dell'immagine: abbiamo visto nel primo paragrafo che questo corrisponde ad aggiungere una costante ad ogni pixel, il che è disponibile attraverso la funzione *ippiAddC.* Proviamo, stavolta, a partire dal prototipo contenuto nel manuale: degli otto casi presentati, il nostro è il sesto.

```
ippiAddC_<mod>(const Ipp<datatype> value[3],
    Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
                        roiSize, int scaleFactor);
```

Il manuale elenca anche una serie di *<mod>* disponibili: quello che fa per noi è *8u_C3IRSfs* (*Sfs* significa che è attivo il *Saturation and Fixed Scaling mode:* niente di cui dobbiamo preoccuparci). Il primo parametro indica un vettore di tre valori (uno per ogni canale) che si vogliono aggiungere, l'ultimo il fattore di *scaling*, che noi lasciamo a 0. Ecco, quindi, l'implementazione dell'handler del pulsante *"+ luminosità":*

```
void CIPPEsempioDlg::OnAddValue()
{
  Ipp8u val[3] = {1,1,1}; // incremento di uno
```

FORUM IPP

Il posto migliore (praticamente l'unico) per postare domande su IPP a gente competente è il Forum Ufficiale: http://softwareforums.intel.com/ids/board?board.id=IPP Il Forum è frequentato dai diretti responsabili e dai programmatori della libreria, in genere molto disponibili e capaci di rispondere anche su aspetti non documentati.

MEMORY LEAKS

Non dimenticate mai di distruggere le varie risorse allocate con GDI e con *ippiMalloc*: le immagini bitmap sono notoriamente grandi, e una svista rischia di trasformarsi rapidamente in tragedia. Date un'occhiata, a tal proposito, alla funzione *CIPPEsempio::DestroyWindow().*

```
    ippiAddC_8u_C3IRSfs(val, img, imgStep, size, 0);
    ImgToBmp();
    Blit();
}
```

Analogo discorso per il pulsante inverso, che fa uso della funzione *ippiSubC.*

FUNZIONI DI FILTRAGGIO

Potremmo ora sbizzarrirci a provare tutte le altre trasformazioni geometriche (rotazione, ridimensionamento, etc…), e aritmetiche (somma di due immagini, operazioni logiche, etc…). Qui, invece, introduciamo un'altra serie di funzioni: le IPP contengono una collezione sterminata di filtri per ogni esigenza, dal *blurring* allo *sharpening,* passando per *Wiener* e *Laplace.* Non abbiamo certo lo spazio per vederli tutti, ma possiamo introdurre gli elementi fondamentali su cui si basano. La stragrande maggioranza dei filtri utilizza un Kernel, ovverosia una matrice ben definita (di solito di 3x3, o 5x5) di valori interi, all'interno della quale viene stabilito un punto d'ancoraggio. Il Kernel viene dunque "passato" sull'immagine in iterazioni successive, facendo in modo che ad ognuna il "punto di ancoraggio" finisca su un pixel differente.
Vengono stabilite delle regole da applicare, utilizzando i valori del Kernel e quelli dei pixel corrispondenti.

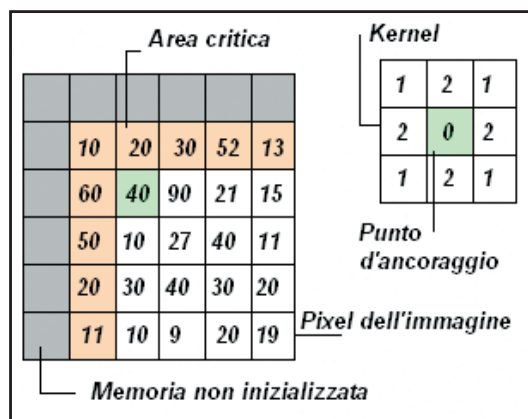



Fig. 6: **Esempio di filtro su un'immagine**

Per capir meglio proviamo a guardare la **Figura 6**, dove ad un'immagine viene applicato un Kernel specifico. Poniamo il punto di ancoraggio sul pixel indicato in verde, e stabiliamo che, come regola, il pixel si ritroverà ad avere la media di tutti i valori dell'intorno, moltiplicati per i rispettivi valori del kernel. In questo caso il valore del pixel diventerà.

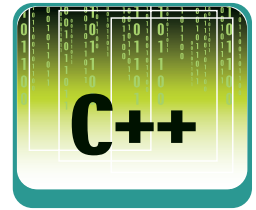
```
(10*1 + 20*2 + 30*1 + 60*2 + 40*0 + 90*2 +
                    50*1 + 10*2 + 32*1)/9 = 53
```

E così via per ogni altro pixel dell'immagine. Tutto questo è piuttosto semplice: l'unico punto cui bisogna fare estrema attenzione sono i bordi. Se si prova ad applicare il filtro ad uno dei pixel segnati in arancione come 'area critica', si dovrà fare i conti con l'area grigia, ovverosia con zone di memoria non inizializzate. Questo conduce nel migliore dei casi a risultati inattesi, e nel peggiore ad un ineluttabile crash dell'applicazione. Occorre quindi "togliere il bordo", diminuendo opportunamente le dimensioni della *ROI* (l'area di applicazione) e passando il puntatore dell'immagine dalla prima zona sicura (in questo caso, dalla posizione *{1,1}*). Proviamo ad applicare il filtro *"box",* che funziona come quello appena descritto, ad eccezione del fatto che il Kernel è composto dal valore *"1"* in ogni sezione.
Questo è il codice.

```
void CIPPEsempioDlg::OnFilterBox()
{
    IppiSize ROI; // Area di applicazione
    ROI.width = size.width - 2; // Togliamo due unità
    ROI.height = size.height - 2; // per eliminare il
                                                  bordo
    IppiSize mask;   // Dimensioni del Kernel
    mask.width = 3; // Impostiamo 3 unità
    mask.height = 3;
    IppiPoint anchor; // Il punto di ancoraggio
    anchor.x = 1; // Lo impostiamo al centro
    anchor.y = 1;
    Ipp8u *start = img + // partiamo dall'immagine
    imgStep + // una riga sotto
    3; // e un pixel (ovvero 3 byte) dopo
    ippiFilterBox_8u_C3IR(start, imgStep, ROI, mask,
                                               anchor);
    ImgToBmp();
    Blit();
}
```

CONCLUSIONI

Lo spazio, purtroppo, è tiranno, ed è impossibile riuscire ad introdurre per intero quella marea di funzioni che è fornita dalla libreria IPP, anche limitandoci esclusivamente al dominio delle immagini. Nel codice d'esempio ho inserito un paio di sorprese che sono sicuro v'interesseranno, in particolar modo ho dato spazio alle funzioni statistiche per il calcolo dell'istogramma delle intensità dei pixel, che presentano anche il principio per il quale si può scegliere un solo canale d'interesse a partire da un'immagine multicanale.
Spero che quest'introduzione vi sia servita come antipasto per l'argomento, e che vi permetta di essere subito produttivi nel magico e rapidissimo mondo delle *Intel Performance Primitives!*

Roberto Allegra

NOTA

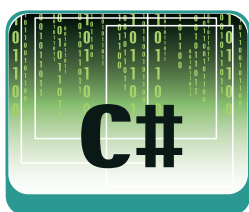
CLASSI WRAPPER

Fra i grandi pregi degli esempi (soprattutto quelli in linea), c'è il fatto che vengono implementate delle classi *wrapper* per orientare la libreria ad oggetti e sfruttare il polimorfismo, che molto si confà alle IPP. Il risultato è molto coerente, anche se si introduce un overhead che pesa sulle prestazioni. Alcuni esempi espongono classi wrapper anche per C#!

CONTATTA L'AUTORE

Per ogni richiesta /critica/suggerimento l'autore può (e deve!) essere contattato all'indirizzo
Roberto.allegra@ioprogrammo.it

Griglia si, ma con classe

Utilizziamo il controllo Property Grid e la reflection per creare un legame stretto fra i dati della nostra applicazione, gli oggetti e le classi che li rappresentano. Un metodo insolito ma molto potente

L'idea è molto semplice: utilizzare una property grid identica a quella di Visual Studio in un nostro progetto. L'utilizzo di un controllo del genere può risultare indicato in molti casi, in tutti quelli in cui è necessario settare le proprietà di un oggetto.
Nel primo esempio che vi proporremo cambieremo il numero di targa di una macchina utilizzando una property grid.
Direte voi, e qual è la novità? Sarebbe stato possibile farlo anche tramite una normale text box, la risposta è che la property grid analizza i componenti di una classe e grazie alla reflection produce in automatico i campi necessari a controllare l'oggetto.

FACCIAMOLO DALL'AMBIENTE

Prima di avventurarci nei meandri del codice, facciamo un esperimento più semplice, utilizzando l'ambiente in modo visuale. Creiamo un nuovo progetto C# di tipo *Windows Application*. Dal menu *tools* clicchiamo su *Add/Remove toolbox items* selezioniamo il componente property *grid* e aggiungiamolo alla toolbox dando *ok.*
Creiamo adesso una classe di prova cliccando con il tasto destro del mouse sul nome dell'applicazione nella finestra di *solution explorer* e aggiungiamo una nuova classe da *Add/Add Class.*
La nostra nuova classe sarà la seguente:

```
public class Automobile
{
   private int cilindrata;
   private string targa;
   private Bitmap foto;

   public int Cilindrata
   {
     get
     {
        return cilindrata;
      }
      set
      {
        cilindrata=value;
      }
   }

   public string Targa
   {
     get
     {
        return targa;
     }
     set
     {
        targa=value;
     }
   }

   public Bitmap Foto
   {
     get
     {
        return foto;
      }
      set
      {
          foto=value;
       }
     }
}
```

ricordiamoci anche di aggiungere la clausola:

```
usingSystem.Drawing;
```

a questo punto non ci resta che trascinare il componente property *grid* sulla form e settare:

Conoscenze richieste
Conoscenze medie di C# e .NET

Software
.NET Framework SDK 1.1

Impegno

Tempo di realizzazione

```
publicForm1()
  {
     //
     // Required for Windows Form Designer support
     //
     InitializeComponent();

     Automobile auto=new Automobile();
     auto.Targa="AA1234ZZ";
     auto.Cilindrata=1298;
     propertyGrid1.SelectedObject=auto;

  }
```

Mandando in esecuzione il programma otterrete una form contenente una griglia in grado di controllare gli oggetti della classe. Aggiungendo una label alla form, possiamo ancora fare qualche passo avanti aggiungendo il seguente codice:

```
private void propertyGrid1_
        SelectedGridItemChanged(object sender,
                     System.Windows.Forms
          .SelectedGridItemChangedEventArgs e)
  {
    label1.Text=e.NewSelection.Value.ToString();
  }

private void propertyGrid1_
              PropertyValueChanged(object s,
                     System.Windows.Forms
             .PropertyValueChangedEventArgs e)
   {
      label1.Text=e.ChangedItem.Value.ToString();

   }
```

Che ci consente di modificare a runtime il contenuto di *label1* semplicemente modificando il contenuto della *property grid.*

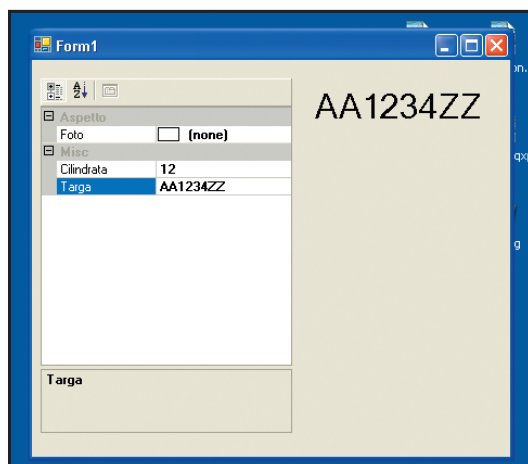



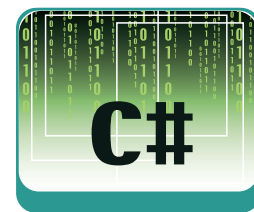
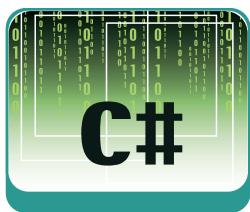
***Fig. 1:* La nostra applicazione si presenta così. Le proprietà dell'oggetto sono perfettamente rappresentate dalla property grid**

DALLA PRATICA ALLA TEORIA

Il controllo *PropertyGrid* si crea e si visualizza su un contenitore come un qualsiasi altro controllo. Una volta creato il controllo *PropertyGrid,* basta impostare la proprietà *SelectedObject,* passandogli come argomento un'istanza di una qualunque classe.
Il funzionamento del controllo *PropertyGrid* è basato, dietro le quinte, sul concetto di *reflection* del codice.
Mediante la *Reflection* il controllo esamina la classe dell'oggetto e scopre quali siano le sue proprietà pubbliche, e quali naturalmente sono i valori da esse assunti.
Le proprietà possono essere suddivise per categoria, cosa per la quale, sempre tramite reflection, vengono ricavati i valori di particolari attributi applicati alle proprietà stesse, attributi che vedremo nel prossimo paragrafo.
Oltre a questa visualizzazione. si ha anche la possibilità di ordinare le proprietà in ordine alfabetico, agendo sui pulsanti della Toolbar in alto. Nella parte bassa, viene invece mostrato un riquadro contenente l'eventuale descrizione.
Naturalmente è possibile personalizzare l'aspetto della *PropertyGrid,* ad esempio impostando a false le proprietà *ToolbarVisible* ed *HelpVisible* si nascondono rispettivamente la toolbar e il pannello di descrizione.
Senza la toolbar è possibile dunque impostare da codice la modalità di ordinamento alfabetico ponendo al valore true la proprietà a *PropertySort-.Alphabetical.*

CATEGORIE DI PROPRIETÀ

Per usufruire della suddivisione in categorie e della possibilità di visualizzare una descrizione per le proprietà contenute in una *PropertyGrid* è necessario apportare delle modifiche al codice della classe, aggiungendo dei particolari attributi alle proprietà.
Innanzitutto occorre aggiungere un'istruzione *using,* per importare il namespace *System.ComponentModel.*
Adesso supponiamo di voler inserire la proprietà Foto della classe *Automobile* in una categoria *Aspetto,* e di voler dare una descrizione alla proprietà stessa. È sufficiente fare uso degli attributi *Category* e *Description* in questa maniera:

```
[Category("Aspetto")]
[Description("La foto dell'automobile")]
public Bitmap Foto
{
```

```
        get
        {
                return foto;
        }
        set
        {
                foto=value;
        }
}
```

La stessa *PropertyGrid* dell'esempio precedente, impostando diverse categorie, apparirebbe ad esempio come nella **Figura 2**.

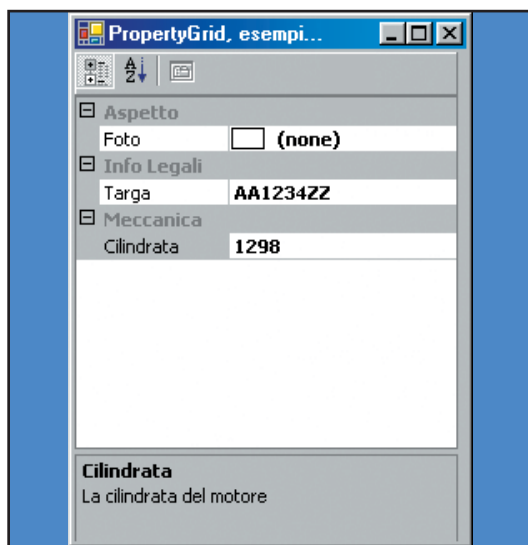



Fig. 2: Categorie di proprietà

Le proprietà per le quali non si specifica alcuna categoria, verranno inserite per default nella categoria *Misc* (miscellaneous).
Poiché come detto la *PropertyGrid* usa la reflection per ottenere le proprietà, ogni proprietà pubblica verrebbe visualizzata, ma nel caso in cui si voglia evitare questo comportamento, per qualche proprietà in particolare, è possibile utilizzare l'attributo *Browsable.*
Ad esempio se volessimo nascondere, alla *PropertyGrid,* la proprietà *Targa,* basta scrivere quest'ultima così:

```
[Browsable(false)]
public string Targa
{
        get
        {
                return targa;
        }
        set
        {
                targa=value;
        }
}
```

Un'altra maniera per filtrare le proprietà da visualizzare in una *PropertyGrid* è quella di utilizzare la sua proprietà *BrowsableAttributes,* specificando le categorie che si vogliono rendere visibili, ad esempio, se *propsGrid* è la nostra *PropertyGrid:*

```
propsGrid.BrowsableAttributes=new
                    AttributeCollection(new Attribute[]
                    { new CategoryAttribute("Name"),
                    new CategoryAttribute("Layout")  }
);
```

In questa maniera, solo le proprietà appartenenti alle categoria *Name* oppure *Layout,* verrebbero visualizzate nella griglia, questo rende molto più semplice il compito di filtrare le proprietà di un oggetto.

TIPI DELLE PROPRIETÀ

Il controllo *PropertyGrid* supporta molteplici tipi in maniera automatica, ad esempio la classe *Automobile* che abbiamo implementato, possiede la proprietà *Foto* di classe *Bitmap.* Nella *PropertyGrid* viene visualizzata una miniatura della Bitmap utilizzata, se presente, e le varie informazioni su di essa, come risoluzione, dimensioni o altro (vedi **Figura 3**).

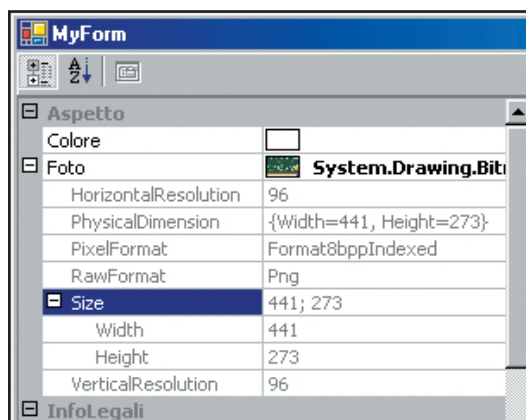
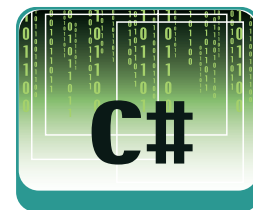



Fig. 3: impostazione di una proprietà Bitmap

Inoltre è possibile sfogliare il file system per cercare l'immagine da impostare, tramite la classica dialog di apertura file. La *PropertyGrid* riconosce in maniera automatica altri tipi di dati e permette di visualizzarli e modificarli con un editor appropriato.
Aggiungiamo ora alla classe *Automobile* altri due campi e relative proprietà:

```
private DateTime dataImmatricolazione;
private Color colore;

[Category("InfoLegali")]
```

```
[Description("La data di immatricolazione
                                  dell'automobile")]
public DateTime DataImmatricolazione
{
   get
   {
      return dataImmatricolazione;
   }
   set
   {
      dataImmatricolazione=value;
   }
}

[Category("Aspetto")]
[Description("Il colore della carrozzeria")]
public Color Colore
{
   get
   {
      return colore;
   }
   set
   {
      colore=value;
   }
}
```

In questo caso la *PropertyGrid* permetterà di selezionare la data di immatricolazione tramite un controllo *DateTimePicker,* ed il colore per mezzo dell'altrettanto classica e intuitiva tavolozza dei colori.

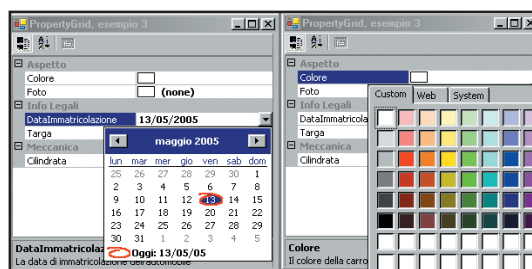



Fig. 4: ***PropertyGrid con DateTimePicker e ColorPicker***

Purtroppo non sono molti i tipi supportati direttamente, ed in particolare per un tipo definito dall'utente, bisognerà gestire le conversioni manualmente, fornendo metodi ad hoc per convertire un tipo da e verso una stringa, o anche delle interfacce grafiche per impostare in maniera rapida i valori di una proprietà di un tipo personalizzato.
Supponiamo di aver creato una nuova classe *Motore*, di cui ogni oggetto *Auto* avrà la propria istanza:

```
public class Motore
{
```

```
      int cilindrata;
      int cavalli;
      int cilindri;
}
```

Se nella classe *Automobile* creiamo il campo ed una proprietà pubblica per impostare il *Motore*, la *PropertyGrid* non riuscirà ad interpretare l'istanza di *Motore*, e visualizzerà come valore della proprietà il nome completo di namespace della classe, ad esempio *MyNamespace.Motore*, ed inoltre non permetterà alcuna modifica.
Quello che c'è da sapere, e che si intuisce dal comportamento attuale, e che il valore della proprietà viene visualizzato dalla *PropertyGrid* ricorrendo al metodo *ToString*, che ogni classe deriva da *Object*. Quindi possiamo scrivere un override di tale metodo in maniera da ottenere nella *PropertyGrid* un'informazione più significativa, ad esempio:

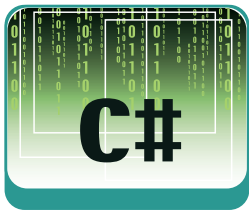
```
public override string ToString()
{
   return String.Format("{0}cc;{1}CV;
                        {2}cil",cilindrata,cavalli,cilindri);
}
```

Un oggetto *Motore* allora potrebbe apparire nella *PropertyGrid* con un valore *"1298 cc;90 CV;4 cil"*.
In questo modo però abbiamo ancora un'informazione di sola lettura, seppur più chiara. Ciò di cui abbiamo bisogno è una procedura che ci permetta di impostare la proprietà, magari ancora nello stesso formato, cioè con tre valori separati da un punto e virgola, in maniera che il primo poi sia preso come il valore della cilindrata, il secondo come il numero di cavalli, e il terzo come il numero di cilindri.
Per ottenere questo risultato è necessario implementare una classe che si occupi della conversione, derivandola da *TypeConverter*, ed implementando i metodi *CanConvertFrom* e *ConvertFrom*.
Il primo si occupa di verificare se una stringa può essere convertita in un dato tipo. Nel nostro caso, vogliamo semplicemente convertire da una stringa, quindi basta verificare appunto che il tipo di origine sia *string*.

```
public class MotoreConverter:TypeConverter
{
   public override bool CanConvertFrom(
     ITypeDescriptorContext context, Type sourceType)
   {
      return sourceType == typeof(string);
   }
}
```

Il metodo *ConvertFrom* invece effettuerà la conversione vera e propria, verificando che la stringa

IL NAMESPACE SYSTEM.COMPONENTMODEL

Il namespace *System.ComponentModel* fornisce le classi usate per implementare il comportamento di controlli e componenti sia runtime che a design-time, cioè all'interno di un ambiente di sviluppo RAD. Contiene ad esempio le classi base per i type converter, per gli attributi, per il databinding e per la gestione del licensing dei controlli.

impostata sia nel formato desiderato.

```
public override object ConvertFrom(
        ITypeDescriptorContext ctx, CultureInfo culture,
                                              object value)
{
    if (value == null)
        return null;
    string sval = value as string;
    if (sval == null)
        throw new NotSupportedException("Tipo non
                                              supportato");
    string[] values = sval.Split(';');
    if (values.Length == 3)
    {
        string str=values[0];
        if(values[0].IndexOf("cc")>0)
            str=str.Substring(0,str.IndexOf("cc"));
        int cc=Convert.ToInt32(str);
        str=values[1];
        if(str.IndexOf("CV")>0)
            str=str.Substring(0,str.IndexOf("CV"));
        int cv=Convert.ToInt32(str);
        str=values[2];
        if(str.IndexOf("cil")>0)
            str=str.Substring(0,str.IndexOf("cil"));
        int cil=Convert.ToInt32(str);
        Motore motore = new Motore(cc,cv,cil);
        return motore;
    }
    else  throw new NotSupportedException("formato
                                              non valido");
}
```

A questo punto per fare in modo che la *PropertyGrid* utilizzi la classe *MotoreConverter* per effettuare la conversione da una stringa, basta applicare alla proprietà *Motore* della classe *Auto* l'attributo *TypeConverter* in questa maniera:

```
[Category("Meccanica")]
[Description("Il motore dell'automobile")]
[TypeConverter(typeof(MotoreConverter))]
public Motore Motore
{
    get
    {
        return motore;
    }
    set
    {
        motore=value;
    }
}
```

Se ora si esegue nuovamente l'applicazione di esempio avremo si avrà la possibilità di impostare la proprietà tramite la *PropertyGrid*, sia dando tre valori separati dal punto e virgola, ad esempio *"1900;150;4"*, oppure nel formato *"1900 cc; 150 CV; 4 cil"*. È sicuramente un formato comodo, ma ciò non ci ripara da eventuali errori di battitura, ad esempio basta scrivere il valore della proprietà come *"1900 cc; 150 CV; 4 ci"*, dimenticando l'ultima lettera, e ci verrà segnalata un'eccezione di valore non valido. Allora un'altra opportunità che possiamo sfruttare è quella di utilizzare delle proprietà innestate, cioè una classe può avere come proprietà degli oggetti che a loro volta sono proprietà di un'altra oggetto. Nel nostro caso la proprietà *Motore* della classe *Auto*, potrebbe avere a sua volta delle proprietà, *Cilindrata, Potenza, NumeroCilindri:*

```
public int Cilindrata
{
    get
    {
        return cilindrata;
    }
    set
    {
        cilindrata=value;
    }
}

public int Potenza
{
    get
    {
        return cavalli;
    }
    set
    {
        cavalli=value;
    }
}

public int NumeroCilindri
{
    get
    {
        return cilindri;
    }
    set
    {
        cilindri=value;
    }
}
```

Con la sola aggiunta delle tre proprietà, la classe *PropertyGrid* può visualizzare la proprietà Motore con un segno + alla sua sinistra, ad indicare la possibilità di poter espandere le sue proprietà innestate. Per abilitare questa modalità è necessario però utilizzare un *TypeConverter* apposito, ad

BIBLIOGRAFIA

• **PROFESSIONAL C#** ***Robinson*** **et al (Wrox Press)**

• **PROGRAMMING WINDOWS WITH C#** ***Petzold*** **(Microsoft Press)**

• **USER INTERFACES IN C#: WINDOWS FORMS AND CUSTOM CONTROLS** ***MacDonald*** **(Apress)**

esempio quello che ci è fornito dal .NET Framework, è la classe *ExpandableObjectConverter.* Provate ad utilizzarla al posto del precedente *TypeConverter* da noi implementato e vedrete una *PropertyGrid* simile a quella della figura seguente. Le tre proprietà innestate sono modificabili singolarmente, ma il valore visualizzato per la proprietà madre *Motore* non si aggiorna automaticamente quando si variano le singole sotto-proprietà. Cosa forse ancor peggiore è che non è possibile editare come nell'esempio precedente il valore della proprietà, con i tre valori separati da punto e virgola, cioè non è più utilizzabile il metodo di parsing della stringa che avevamo implementato.

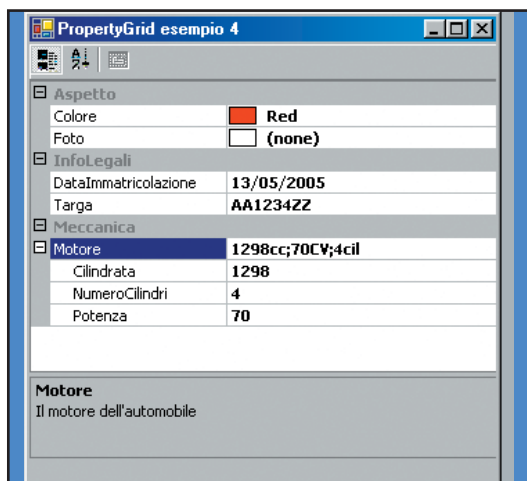



Fig. 5: ***Visualizzazione di proprietà innestate.***

Il primo problema, quello del *refresh* immediato, si risolve facilmente utilizzando ancora una volta un attributo, *RefreshProperties,* da applicare alle singole proprietà innestate, ad esempio per la proprietà *Cilindrata* della classe *Motore* scriveremo:

```
[RefreshProperties(RefreshProperties.Repaint)]
public int Cilindrata
{
   get
   {
      return cilindrata;
   }
   set
   {
      cilindrata=value;
   }
}
```

In questa maniera la *PropertyGrid* aggiornerà anche la proprietà *Motore* al variare di una delle sottoproprietà. Il problema della modifica diretta della proprietà *Motore,* cioè della proprietà madre di altre sottoproprietà, si risolve in maniera un po' meno immediata, ma non troppo complessa. Pensandoci bene quello che vogliamo è utilizzare un *TypeConverter* che utilizzi i metodi di *MotoreConverter* per il parsing delle stringhe, ma allo stesso tempo che permetta di espandere le sottoproprietà come un *ExpandableObjectConverter.* Forse la prima cosa che viene in mente è proprio quella esatta: basta derivare la classe *MotoreConverter* da *ExpandableObjectConverter.*

```
public class MotoreConverter:
                        ExpandableObjectConverter
{
//corpo della classe...
}
```

A questo punto abbiamo una proprietà editabile direttamente con un nostro metodo di parsing personalizzato, oppure con la facoltà di editare le singole sottoproprietà ed in maniera che i cambiamenti si riflettano immediatamente sul valore della proprietà madre.

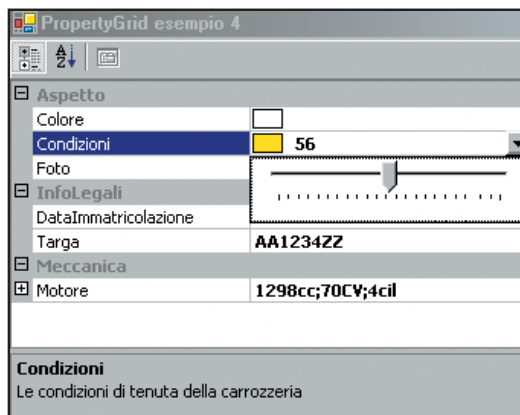
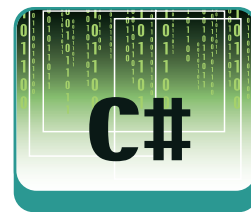



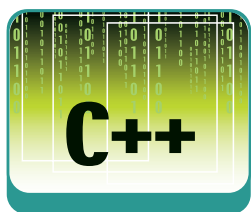
Fig. 6: ***Una schermata dell'applicazione con controllo personalizzato***

CONCLUSIONI

Il controllo *property grid* può rivelarsi utile in un moltissimi casi. Sicuramente in tutti i casi in cui è necessario fornire all'utente un'interfaccia amichevole per la gestione dei dati. Il suo punto di forza è la reflection che ci consente di adattarlo alle nostre classi con uno sforzo assolutamente minimo. Vero è che non tutti i tipi di dati sono gestiti, ed altrettanto vero è che non ci sono editor personalizzati per l'immissione di dati complessi, ma con un minimo di sforzo si riesce comunque a creare un proprio editor derivando direttamente dalla classe *System.Drawing.Design.UITypeEditor.* Infine il controllo è esattamente lo stesso che viene utilizzato come editor delle proprietà in Visual Studio, garantisce pertando una certa omogeneità con l'intero sistema. Sicuramente si tratta di una tecnica da provare.

Antonio Pelleriti

Potete rivolgere domande di chiarimenti o ulteriori richieste all'autore all'indirizzo antonio.pelleriti@ioprogrammo.it, o ancora sul forum di ioProgrammo (forum.ioprogrammo.net) o sul sito www.dotnetarchitects.it.

Salviamo la Configurazione

Impariamo ad usare il Configuration Application Block di Microsoft per risolvere una volta per tutte il problema di rendere persistenti i dati di configurazione delle nostre applicazioni

Non esiste applicazione più complessa di "hello world" che non abbia bisogno di salvare opzioni di configurazione. Più di una volta vi sarete scontrati con il problema di dover rendere persistenti dati come stringhe di connessione o esotiche opzioni di visualizzazione scelte dall'utente, ogni sviluppatore prima o poi ha dovuto inventarsi un modo per salvare e rileggere le preferenze in un qualche formato, dai file *.ini* ai file XML a record in un DataBase. Questa scelta dipende dalla tipologia di dati da rendere persistente e la pulizia del codice sviluppato dipende dalla sensibilità del programmatore.

GLI APPLICATION BLOCKS

Alcune funzionalità dei nostri programmi si ripetono immutate. Specialmente per chi sviluppa in maniera professionale si pone il problema di non reinventare la ruota ad ogni nuovo progetto: il riutilizzo di codice e di conoscenze è uno dei principali mezzi per ottenere qualità ed economicità nello sviluppo. In quest'ottica Microsoft, nell'ambito dell'iniziativa patterns and practices, ha sviluppato e fornito agli sviluppatori gli *Enterprise Library Application Blocks*, in cui sono individuati alcuni dei più comuni sottoinsiemi di funzionalità che gli sviluppatori si trovano ad affrontare. Ogni *Application Block* risolve un problema, o meglio può essere utilizzato come base per lo sviluppo di macrofunzionalità nelle proprie applicazioni. In dettaglio, nella ultima versione disponibile, gli *Application Blocks* sono:

- **Caching Application Block** Permette di creare una cache locale.
- **Configuration Application Block** Permette di leggere e scrivere informazioni di configurazione su vari supporti.
- **Data Access Application Block** Fornisce le più comuni funzionalità di accesso ai dati.
- **Cryptography Application Block** Permette di aggiungere funzionalità crittografiche alle proprie applicazioni.
- **Exception Handling Application Block** Fornisce le basi per una gestione accurata e completa delle eccezioni.
- **Logging and Instrumentation Application Block** Permette l'aggiunta in modo semplice di funzionalità di *Log* e instrumentazione del codice.
- **Security Application Block** fornisce una infrastruttura di autenticazione e autorizzazione.

È disponibile da marzo anche l'*Updater Application Block*, che permette di creare un framework per l'aggiornamento del proprio software. Questi componenti software, disponibili per il download gratuito, vengono forniti sia come assembly compilati e riutilizzabili sia come codice sorgente consultabile e modificabile (leggere attentamente le note di copyright installate col software).

IL CONFIGURATION APPLICATION BLOCK

Il *Configuration Application Block* automatizza in modo pulito e semplice la gestione della persi-

COME INIZIARE

È necessario scaricare e installare dal sito http://msdn.microsoft.com/library/default.asp?url=/library/ le Enterprise Library, al momento in cui scriviamo la versione più recente è relativa a Gennaio 2005. La fase di installazione potrebbe richiedere tempi leggermente lunghi, non preoccupatevi se tutto sembra bloccato per qualche minuto.

stenza delle opzioni. I vantaggi nell'uso di questo strumento vanno dalla certezza di usare un codice pulito sviluppato da Microsoft, al poter riutilizzare sempre lo stesso codice in ogni nostra applicazione, alla gestione trasparente dei formati di salvataggio, fall'ini all XML. C'è di più: è possibile scrivere le proprie classi che intervengono nel percorso di serializzazione, permettendo la massima libertà allo sviluppatore nel decidere i dettagli del salvataggio. La struttura funzionale del *Configuration Application Block* è quella riportata in **Figura 1**, dove sono evidenziati due oggetti all'interno del *Block*.

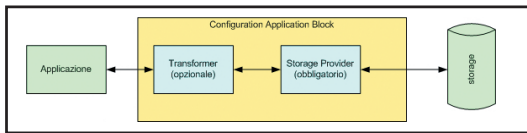



***Fig. 1:* Struttura funzionale**

L'applicazione fornisce al *Configuration Application Block* un oggetto che contiene le opzioni da salvare. Questo oggetto viene processato da due stadi. Il primo, opzionale, è il *Transformer* e serve a trasformare, qualora ce ne fosse bisogno, l'oggetto da persistere in modo che il blocco di *StorageProvider* sappia come salvare/leggere i dati dallo storage. Immaginiamo per esempio di avere un'applicazione che può fornire un'istanza di un *Dictionary,* e uno *StorageProvider* che può ricevere in ingresso solo *HashTable:* il *Transformer* si occuperà di fare la trasformazione, e lo *StorageProvider* saprà come scrivere l'*HashTable* sul dispositivo di storage, per esempio un *DataBase* o un file di testo. L'inverso avverrà in lettura. La libreria permette di scrivere *Transformer* e *StorageProvider* personalizzati, in modo da poter coprire i vari casi, formati, opzioni di storage che si potrebbero presentare allo sviluppatore. Il *Configuration Application Block* fornisce comunque un meccanismo preimpostato per poter da subito cominciare a sfruttarne le potenzialità: si tratta dei blocchi di utilizzo del formato XML. Come si vede in **Figura 2** la struttura generale rimane invariata, fatta eccezione per i due componenti che sono l'*XMLSerializerTransformer* e l'*XMLFileStorageProvider* e per lo storage che è fornito da uno o più file XML.

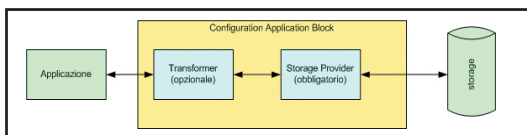



***Fig. 2:* Struttura funzionale per XML**

Nel nostro esempio utilizzeremo il formato standard XML, in modo da poterci concentrare in pratica sugli aspetti di base che permettono comunque un utilizzo soddisfacente e completo della funzionalità. Nell'ambito dello sforzo della *Enterprise Library* di suggerire pratiche di programmazione più "pulite" nel loro orientamento agli oggetti, il salvataggio e la lettura dei dati avviene "a oggetti", nel modo più semplice e naturale possibile: si crea una classe e si utilizza un'istanza di tale classe da dare in pasto al *Configuration Manager,* in lettura o in scrittura. Et voilà, la classe sarà rispettivamente popolata coi valori di configurazione o salvata nel file scelto.

TUTTO IN UNA CLASSE

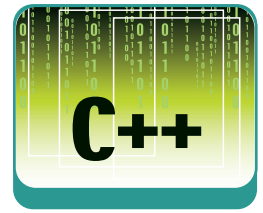
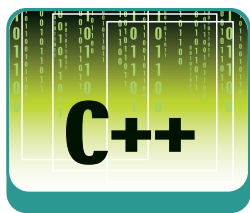
Come già anticipato, il meccanismo di utilizzo del *Configuration Application Block* è estremamente semplice. Tutto comincia individuando i parametri di configurazione che vogliamo salvare. Creeremo poi una classe per ogni sezione di configurazione, e aggiungeremo le opzioni da salvare come proprietà. Istanze di questa classe ci serviranno per leggere e scrivere le nostre opzioni. Per far questo creeremo un oggetto e lo passeremo ad uno dei due metodi statici di lettura o scrittura della classe *Microsoft.Practices.EnterpriseLibrary.Configuration.ConfigurationManager,* ovvero

```
– public static void WriteConfiguration(string
                    sectionName,   object configValue)
– public static object GetConfiguration(string
                                        sectionName)
```

dove *sectionName* è una stringa contenente il nome della sezione di configurazione del nostro file, e *configValue* è l'oggetto che vogliamo serializzare nel file stesso. Un'istanza di questo file verrà restituita come *object* da *GetConfiguration().*

FACCIAMO UN ESEMPIO

Scriviamo un semplice programmino *WinForms* in C# che illustri il funzionamento di quanto abbiamo appena detto. Utilizzeremo VisualStudio 2003, ma quello che diremo può essere fatto con qualunque sistema di sviluppo. Il nostro programma utilizzerà i servizi del *Configuration Block* per salvare le opzioni riguardanti la posizione e la larghezza della sua finestra principale. Per prima cosa creiamo una nuova applicazione *WinForms,* e aggiungiamo tra i riferimenti la *dll:*

```
C:\Programmi\Microsoft Enterprise Library
          \bin\Microsoft.Practices.EnterpriseLibrary
                                .Configuration.dll
```

Utilizza questo spazio per le tue annotazioni

Aggiunto il riferimento, andiamo ad aggiungere il namespace all'inizio del nostro sorgente:

```
using Microsoft.Practices.EnterpriseLibrary
                                    .Configuration;
```

Creiamo una classe che esponga come proprietà le opzioni di configurazione e che chiameremo *ConfigClass* (si noti che qualunque nome andrebbe bene!):

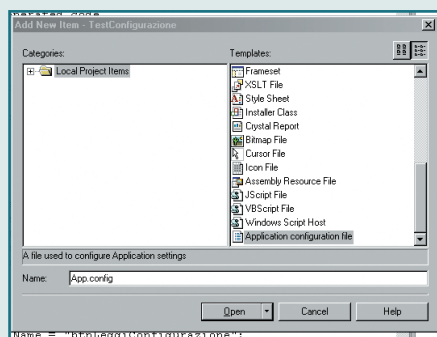
```
using System;
namespace TestConfigurazione
{
   public class ConfigClass
   {
      int _Top;
      int _Left;
      int _Width;
      public int Top
      {
         get{return _Top;}
         set{_Top = value;}
      }
      public int Left
      {
         get{return _Left;}
         set{_Left = value;}
      }
      public int Width
      {
         get{return _Width;}
         set{_Width = value;}
      }
   }
}
```

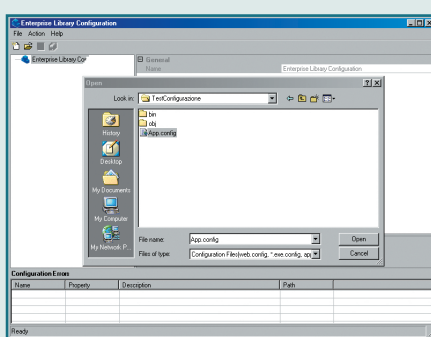
L'ENTERPRISE LIBRARY CONFIGURATION

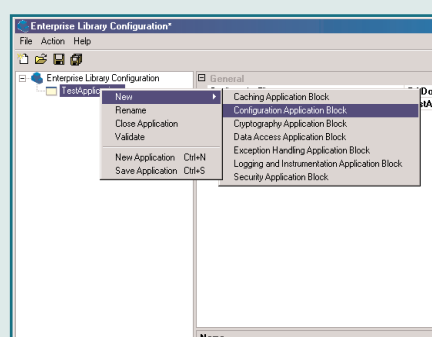
Tutti gli *Enterprise Library Application Blocks* leggono le proprie impostazioni da file di configurazione XML, il principale dei quali è proprio l'*app.config*, ovvero il file principale di configurazione dell'applicazione WinForms. Il nome di questo file viene cambiato da VisualStudio ad ogni *build* da *app.config* a quello dell'eseguibile cui fa riferimento (es.: *TestConfigurazione.exe*) cui viene appesa l'estensione *.config* (es.: *TestConfigurazione.exe.config*).
Le impostazioni dei vari *Application Blocks* sono piuttosto complesse, soprattutto facendo riferimento gli uni agli altri, e sarebbe difficile inserirle a mano. Ci viene in aiuto una applicazione distribuita con l'Enterprise Library che graficamente ci aiuta a costruire il blocco di configurazione necessario e, se del caso, ad aggiungerlo all'*app.config* del nostro progetto.
Tale applicazione è l'*Enterprise Library Configuration*, richiamabile dal menu *Avvio->Programmi->Microsoft Patterns and Practices->Enterprise Library*, applicazione di cui è visibile una schermata in **Figura 3**.

CONFIGURIAMO LA CONFIGURAZIONE!

Il Configuration Block legge le proprie opzioni di configurazione da un file generale che corrisponde allo stesso file di configurazione utilizzato dall'applicazione. Scopriamo come fare a realizzare facilmente questo file

> LE OPZIONI INIZIALI

1 Aggiungiamo al nostro progetto un App.Config. Per farlo clicchiamo con il tasto destro del mouse nel solution explorer, poi su Add New Item e infine scegliamo Application Configuration File

> IMPORTARE IL FILE

2 Dopo aver fatto partire il programma Enterprise Library Configuration, dal menu Avvio->Programmi->Microsoft Patterns and Practices->Enterprise Library, importiamo il file App.Config appena creato.

> CONFIGURATION BLOCK

3 Dopo aver rinominato l'Application in TestConfigurazione, aggiungiamo un blocco Configuration, cliccando con il pulsante destro del mouse sull'etichetta dell'applicazione TestConfigurazione dell'albero di sinistra.

Ogni applicazione che andremo a configurare è un nodo *Application* dell'albero di sinistra. Nel nostro esempio andremo a scrivere le nostre configurazioni su un apposito file XML.
Nel caso in cui invece abbiate già delle opzioni sul vostro *App.config* nessun problema: è possibile importare il file esistente nell'applicazione di configurazione, e mantenerlo sincronizzato.

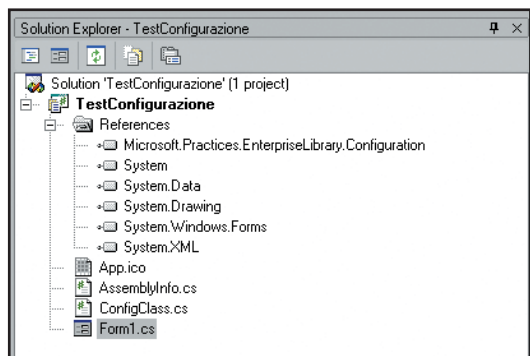



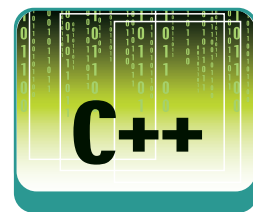
Fig. 3: ***Enterprise Library Configuration***

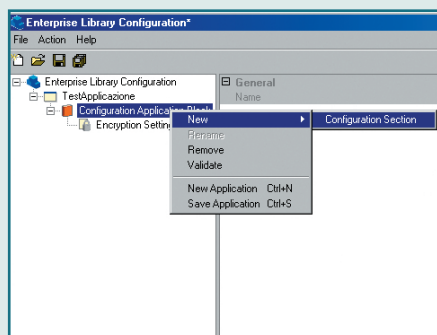
Vediamo che il nodo *Configuration* ha un sottonodo che fa riferimento ai servizi di crittografia. Se volessimo sfruttare il *Cryptography Application Block* per nascondere ad occhi indiscreti i file di configurazione dovremmo valorizzare questa opzione, che però è al di fuori degli argomenti trattati in questo articolo.
Possiamo definire quante sezioni di configurazione vogliamo, sullo stesso file XML o su file differenti. Per la nostra applicazione utilizzeremo il *Configuration Application Block*, lo doteremo con una sezione chiamata *OpzioniGenerali*, che a sua volta sarà dotata di XML *File Storage Provider* e di un *XML Serializer Transformer.*
La proprietà file dello storage provider sarà settata ad opzioni *.config*. Riepilogando nella stessa directory che contiene l'eseguibile dovremo inserire un *nomeprogramma.exe.config* e un *opzioni.config*. Il primo conterrà il file di configurazione per il *configuration block*, il secondo invece conterrà le opzioni che vorremo salvare relative al nostro programma. Come già detto il programma si aspetterà di trovare il file *opzioni.config* nella stessa directory dell'eseguibile. Le possibilità sono due: o copiamo a mano il file nella cartella di produzione (*Debug* o *Release* che sia) o scriviamo una semplice riga di comando nell'evento *PostBuildEvent* del progetto che faccia la copia del file nella cartella di destinazione dopo ogni *build* terminata correttamente.
Per fare questo dovremmo fare click col pulsante destro del mouse sul nome del progetto nel *SolutionExplorer*, scegliere *Build Events*, e inserire in *Post-Build Event Command Line* un *command* del genere:

```
Copy "$(ProjectDir)\opzioni.config"
                    "$(TargetDir)\opzioni.config"
```

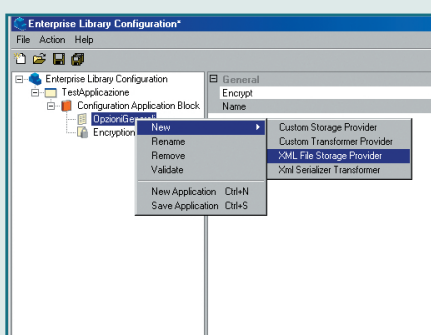
Questo è necessario poiché il *ConfigurationManager* non crea il file di configurazione se non lo trova, ma solleva un'eccezione. Si noti però che utilizzando questa tecnica il file *opzioni.config* viene sovrascritto ad ogni *build*.

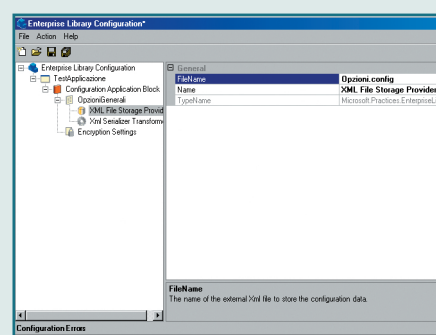
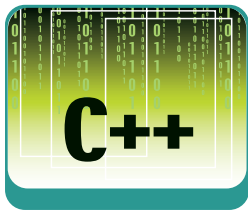
LEGGERE E SCRIVERE

A questo punto siamo pronti per andare a leggere e scrivere istanze di *ConfigClass* nel file di configurazione e nel blocco di configurazione scelto.
Aggiungiamo un membro privato alla classe

MS Pattern and Practice
http://msdn.microsoft.com/library/en-us/dnanchor/html/Anch_EntDevAppArchPatPrac.asp

Enterprise Library Application Block
http://msdn.microsoft.com/library/en-us/dnpag2/html/entlib.asp

> LE SEZIONI

4 Definiamo una sezione di configurazione che chiameremo opzioni generali, ancora una volta facendo click col pulsante destro del mouse, stavolta sul blocco di configurazione appena aggiunto.

> COME SALVARE

5 Per ogni blocco di configurazione specifichiamo sia il modulo che si occuperà di immagazzinare i dati *(Storage)* sia quello che li serializzerà *(Serializer Transformer).* in entrambi i casi scegliamo XML.

> DOVE SALVARE

6 Avendo scelto un file XML come repository delle opzioni di configurazione, dovremo specificare il nome del file. Lo faremo nella casella *FileName* della property grid relativa all'*XML File Storage Provider.*

Form1 (la finestra principale della nostra applicazione), di tipo *ConfigClass*.
Questo oggetto conterrà le opzioni di configurazione. Inizializzeremo questo oggetto nel costruttore:

```
private ConfigClass Cfg;
  public Form1()
  {
    InitializeComponent();
    Cfg= new ConfigClass();
  }
```

Creiamo due *Button*, *btnScriviConfigurazione* e *btnLeggiConfigurazione*, alla cui pressione andremo a salvare o leggere le opzioni.
Per cominciare, possiamo andare a scrivere, nel gestore d'evento del *Click* sul button *btnScriviConfigurazione*, il seguente codice.

```
private void btnScriviConfigurazione_Click(object
                                sender, System.EventArgs e)
{
  Cfg.Top = this.Top;
  Cfg.Left= this.Left;
  Cfg.Width= this.Width;
    ConfigurationManager.WriteConfiguration
                         ("OpzioniGenerali",Cfg);
}
```

Se compiliamo ed eseguiamo l'applicazione e facciamo scrivere la configurazione ecco cosa sarà stato scritto nel file *Opzioni.config* (ovviamente i valori possono differire):

```
<?xml version="1.0" encoding="utf-8"?>
<OpzioniGenerali>
  <xmlSerializerSection type=
  "TestConfigurazione.ConfigClass, TestConfigurazione,
             Version=1.0.1955.19224, Culture=neutral,
                             PublicKeyToken=null">
    <ConfigClass xmlns:xsd="http://www.w3.org
    /2001/XMLSchema" xmlns:xsi="http://www.w3.org
                       /2001/XMLSchema-instance">
      <Top>154</Top>
      <Left>154</Left>
      <Width>560</Width>
    </ConfigClass>
  </xmlSerializerSection>
</OpzioniGenerali>
```

analogamente per estrarre tali dati, ed assegnarli ai valori della finestra, basterà scrivere:

```
private void btnLeggiConfigurazione_Click(object
                                sender, System.EventArgs e)
{
  Cfg= ConfigurationManager.GetConfiguration(
                  "OpzioniGenerali") as ConfigClass;
  this.Top=Cfg.Top;
  this.Left=Cfg.Left;
  this.Width=Cfg.Width ;
}
```

nel gestore d'evento del click dell'altro button.
Spesso quando vengono cambiate le opzioni di configurazione, è opportuno propagare l'informazione ad altri moduli del programma. Per esempio, se l'utente decide di cambiare il colore di sfondo di tutte le finestre, è utile poterne avere notizia all'interno del programma.
A tale scopo è possibile associare un gestore all'evento che scatta quando è stata cambiata la configurazione, ovvero *ConfigurationManager.ConfigurationChanged*. Per esempio, potremmo aggiungere la seguente riga nel costruttore della *Form1*

```
ConfigurationManager.ConfigurationChanged+=new
 ConfigurationChangedEventHandler(ConfigurationMan
                       ager_ConfigurationChanged);
```

così che venga chiamato il metodo *ConfigurationManager_ConfigurationChanged* ad ogni cambiamento:

```
private void ConfigurationManager_
            ConfigurationChanged(object sender,
              ConfigurationChangedEventArgs e)
{
  MessageBox.Show("configurazione cambiata in
    ("+e.ConfigurationFile +"/"+e.SectionName+").");
}
```

A CHE PUNTO SIAMO?

Abbiamo visto come salvare e rileggere in maniera molto semplice opzioni di configurazione indipendenti dall'input dell'utente. Generalmente nelle nostre applicazioni abbiamo la necessità di salvare anche opzioni che vengono selezionate dall'utente. Vedremo nella seconda parte di questo articolo come utilizzare un controllo, il propertygrid che ci permetterà di automatizzare l'input delle opzioni, fornendo nel contempo un look funzionale e professionale alla nostra applicazione.
Vedremo anche tecniche più avanzate per scegliere se mostrare o meno all'utente le opzioni da modificare, sia a design time, sia a runtime. Unendo queste tecniche con quelle viste oggi, avremo un framework potente e di semplice uso da riutilizzare nelle nostre applicazioni. Senza dover reinventare la ruota ogni volta…

Marco Poponi

L'AUTORE

Marco Poponi è laureato in Ingegneria Elettronica. Insegna Lab di Programmazione alla facoltà di Ingegneria dell'Università degli Studi di Perugia. È tra i fondatori di *InnovActive Engineering* (http://dotnet.innovactive.it), azienda di sviluppo software e consulenza specializzata nella tecnologia .NET. Si occupa di progettazione, sviluppo e formazione.

NOTA

In un'applicazione reale, qualora volessimo salvare e ripristinare la posizione della finestra ad ogni chiusura /apertura del programma, dovremmo chiamare questo codice rispettivamente in occasione dell'apertura e della chiusura della finestra: nella nostra demo lo scriviamo nei gestori d'evento dei Click per controllare più agevolmente l'effetto prodotto sul file di configurazione.

Visual Basic.NET ecco il suo Menu!

La barra dei menu è un classico che permette di organizzare in modo efficace tutte le funzioni offerte all'utente. Vediamo come fare per dotare le nostre applicazioni di questa funzionalità

La disposizione e la programmazione dei menu costituisce una componente essenziale nello studio dell'interfaccia utente essa viene giudicata valida da quanto: facilmente, rapidamente e logicamente si trovano le funzioni necessarie all'interno dei menu. In questo articolo, vedremo come progettare una barra dei menu, utilizzando il controllo *MainMenu*, e come realizzare il menu pop-up, adottato per far comparire specifici sottomenu contestuali ad un oggetto, utilizzando il controllo *ContextMenu*.

NOMI DEI MENU

Il primo passo sarà, ovviamente, trascinare un componente *MainMenu* nella form dell'applicazione. Vedremo immediatamente materializzarsi una barra di menu per la nostra applicazione. Ogni nuova voce di menu rappresenta un oggetto *MenuItem* a cui viene attribuito un nome del tipo *Menultem1, Menultem2* e così via, a cui si può fare riferimento all'interno del codice.
Così come abbiamo visto per gli altri controlli utilizzati finora, è importante modificare i nomi predefiniti assegnando dei nomi che abbiano una certa logica, in modo da individuare facilmente l'azione che deve compiere il menu. Possiamo far precedere i nomi dal prefisso *mnu* in modo da identificare facilmente un oggetto come menu.
Ad Esempio

- *mnuFile*
- *mnuModifica*
- *mnuFinestra*

È consigliabile inserire nei nomi di menu, l'intera gerarchia, così per i menu: *Apri, Salva, Chiudi* che compaiono nel menu *File,* i nomi dovrebbero essere: *mnuFileApri, mnuFileSalva, mnuFileChiudi.*
Per modificare il nome di un menu, abbiamo due possibilità:

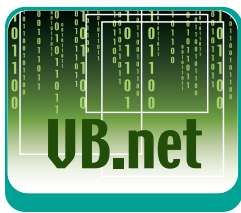
- Visualizzare la finestra delle proprietà e modificare la proprietà *Name.*
- Cliccare con il tasto destro del mouse in un punto della barra dei menu e selezionare la voce *Modifica Nomi* dal menu a discesa, per entrare in modalità *editor dei nomi.* Nella modalità *editor dei nomi* è possibile modificare soltanto i nomi dei menu e non le relative etichette. Per uscire da questa modalità, si deve cliccare con il pulsante destro all'interno della barra dei menu e scegliere nuovamente *Modifica nomi.*

SPOSTARE O COPIARE COMPLETAMENTE LA STRUTTURA DI UN MENU

In Visual Basic .Net 2003 è possibile spostare o copiare completamente un menu (con tutti i suoi sottomenu), all'interno della struttura. Sono infatti supportate, le classiche operazioni di drag-and-drop e di copia ed incolla. Per spostare un menu con il drag-and-drop dobbiamo:

- Selezionare la voce di menu che si vuole spostare.
- Tenere premuto il pulsante sinistro del mouse e trascinare la voce nella nuova posizione.

COMPONENTI INVISIBILI

Un controllo che non è visibile in fase di esecuzione, detto anche componente, fornisce funzionalità all'applicazione. A differenza di altri controlli, i componenti non forniscono un'interfaccia utente e non è quindi necessario visualizzarli nell'area di progettazione Windows Form. Quando un componente viene aggiunto ad una form, viene visualizzato in un'area ridimensionabile nella parte inferiore della finestra. Una volta aggiunto un controllo al contenitore dei componenti, è possibile selezionare il componente ed impostarne le proprietà come per qualsiasi altro controllo della form.

I controlli MainMenu e ContextMenu sono presenti nella Casella degli strumenti che mostra l'elenco dei controlli che si possono trascinare sulle Windows Form. Questi controlli sono contenuti nella libreria System.Windows.Form s.dll che include i controlli (46) che si usano maggiormente

Ad ogni menu di livello principale possono essere associati fino a cinque livelli di sottomenu nidificati. È in ogni modo consigliabile non utilizzare più di tre livelli di sottomenu, per evitare di confondere l'utente.

- Rilasciare il pulsante sinistro del mouse.

Per copiare un menu con le operazioni di copia ed incolla dobbiamo:

- Selezionare la voce o le voci di menu che si desidera duplicare e fare clic su di esse con il pulsante destro del mouse. Dal menu a discesa selezionare la voce *Copia*.
- Selezionare il punto in cui si desidera inserire le voci di menu duplicate.
- Fare clic con il pulsante destro del mouse, e dal menu a discesa selezionare *Incolla* per inserire le voci di menu duplicate.

DISEGNARE UN MENU

Per caratterizzare la barra dei menu, in modo da renderla simile ai menu delle applicazioni Windows, è possibile aggiungere alcuni elementi:

- **Le barre di separazione** Sono le linee di separazione che dividono sottomenu legati logicamente fra loro consentendo una lettura più agevole del menu stesso.
- **I tasti di scelta rapida** Sono comandi da tastiera assegnati alle singole voci di menu, in modo da dare all'utente la possibilità di selezionare un comando senza passare dal sistema dei menu.
- **I tasti di accesso** I tasti di accesso consentono di passare da un menu all'altro direttamente dalla tastiera, premendo *ALT* ed il tasto di accesso sottolineato, per selezionare la voce di menu desiderata.

TASTI DI SCELTA RAPIDA

Per assegnare una combinazione di tasti di scelta rapida ad una voce di menu si deve utilizzare la proprietà *Shortcut*. Se Clicchiamo sulla freccetta posta a destra dell'etichetta *Shortcut,* vengono visualizzati le combinazioni di caratteri di scelta rapida, sono presenti anche i tasti funzione. Dopo aver selezionato la sequenza di tasti (presupponendo il valore di default *ShowShortcut=True*), essi verranno visualizzati sul menu accanto al nome, in modo che sia chiaro per l'utente quali tasti si possono usare per accedere rapidamente ai menu. Non è possibile assegnare gli stessi tasti di scelta rapida a voci di menu diverse.

TASTI DI ACCESSO

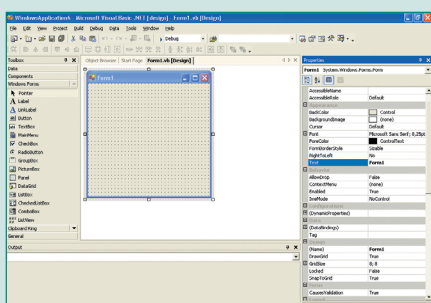
Ad ogni voce di menu, si può associare un tasto di scelta univoco per il suo livello. Quando l'utente, tramite tastiera, preme la combinazione di tasti: *ALT + Tasto di scelta* si ottiene l'effetto della selezione della voce di menu, attivando l'evento *Click* corrispondente. Per associare un tasto di scelta ad un menu, è sufficiente inserire il carattere *"&"* (e commerciale) prima della lettera che si desidera sottolineare quale tasto di scelta. Si preferisce utilizzare la prima lettera del titolo, a patto che la lettera in questione non sia stata utilizzata allo stesso livello.

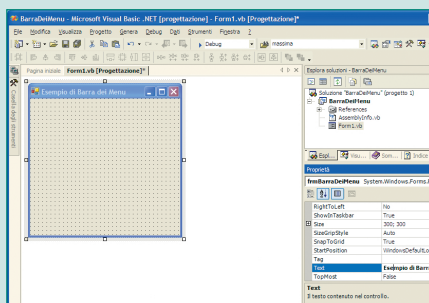
LE PROPRIETÀ DI MENUITEM

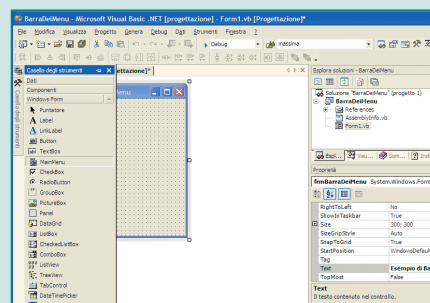
La classe *MenuItem* fornisce proprietà che permettono di configurare l'aspetto e la funzionalità di una

REALIZZARE LA PRIMA BARRA DEI MENU

Un semplice esempio che mostra come dotare un'applicazione di un menu

> UN NUOVO PROGETTO

1 Apriamo *Creiamo un nuovo progetto*, che chiameremo *BarraDeiMenu.* Selezioniamo la finestra *Form1* e, se non è già visualizzata, apriamo la finestra delle proprietà. Il nostro scopo sarà dotare la form in questione di un classico menu.

> LE PROPRIETÀ DELLA FORM

2 Dalla finestra delle proprietà selezioniamo la proprietà *Name* e cambiamo subito il nome in: *frmBarraDeiMenu.* Selezioniamo la proprietà *Text* e modifichiamo il testo visualizzato nella barra del titolo in: *Esempio di Barra dei Menu.*

> AGGIUNGIAMO UN MENU

3 Selezioniamo il controllo *MainMenu* dalla casella degli strumenti (nella sezione *Windows Form*) e trasciniamolo sulla form (otteniamo lo stesso effetto se facciamo doppio clic sull'icona del controllo). Buona parte del lavoro è fatto!

voce di menu. Visualizzando la finestra delle proprietà di una qualsiasi voce di menu, possiamo scorgere tutte le proprietà a disposizione, tra queste:

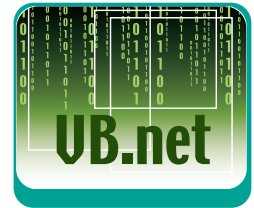
- **Name** consente di specificare il nome interno del menu, vale a dire il nome che utilizzerà lo sviluppatore all'interno del codice.
- **Checked posta a True** visualizza un segno di spunta accanto al testo della voce di menu.
- **RadioChecked posta a True** visualizza un pulsante di opzione anziché un segno di spunta, quando si seleziona una voce di menu.
- **DefaultItem** indica se la voce di menu è quella predefinita.
- **Enabled** indica se la voce di menu è abilitata.
- **Visible** indica se la voce di menu è visibile.
- **MdiList posta a True** popola il menu con l'elenco di finestre figlie visualizzate all'interno della form
- **MergeOrder** indica la posizione relativa della voce di menu quando viene unito ad un altro menu.
- **MergeType** indica il comportamento della voce di menu quando viene unito ad un altro. Potrebbe essere aggiunta, rimpiazzata o rimossa.
- **OwnerDraw** indica se la voce di menu viene creata da codice e non da Windows
- **Shortcut** permette di assegnare alla voce di menu una combinazione di tasti, in modo da dare all'utente la possibilità di selezionare un comando senza passare dal sistema dei menu.
- **ShowShortcut** indica se la combinazione di tasti di scelta rapida associato alla voce di menu debba essere visualizzata alla desta del testo della voce di menu.
- **Text** indica il testo mostrato nella voce di menu.

In particolare, la proprietà *Checked* permette di visualizzare un segno di spunta accanto alla voce di menu per indicare se la funzionalità è attivata o meno. Ne abbiamo un esempio in VB.Net selezionando la voce Visualizza‡Barre degli strumenti, in cui apparirà il sottomenu con un segno di spunta sulle barre degli strumenti attualmente visualizzate nell'ambiente di sviluppo. Per visualizzare il segno di spunta, senza passare dalla finestra delle proprietà, possiamo selezionare una voce di menu (che non sia al livello principale) e cliccare nell'area posta a sinistra della voce. Per cancellare un segno di spunta da codice, possiamo scrivere:

```
mnuBarraStandard.Checked = False
```

ovviamente ponendo la proprietà *Checked =True* il segno di spunta verrà visualizzato di nuovo. La proprietà *Enabled* può essere adottata per impedire all'utente di selezionare un comando non disponibile al momento, pensiamo al comando Incolla che, ovviamente, non può essere utilizzato se non si è prima usato il comando *Copia*. Se un menu non è abilitato viene visualizzato in grigio, in modo da comunicare visivamente all'utente che non è possibile utilizzarlo.
Per attivare o disattivare un menu da codice:

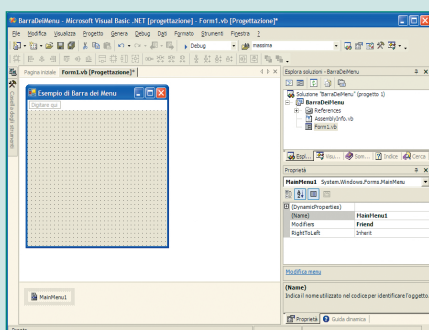
```
MnuModificaIncolla.Enabled = True
MnuModificaIncolla.Enabled = False
```

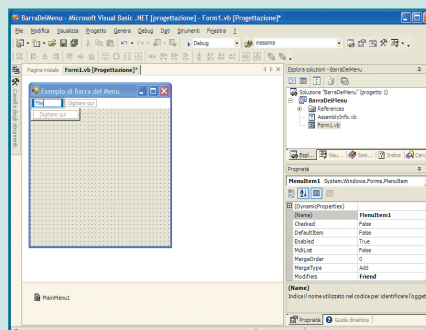
Piuttosto che disattivare una voce di menu può essere preferibile evitare la sua visualizzazione, a questo scopo si utilizza la proprietà *Visible*. Rendendo invisibile una voce di menu, automaticamente si rendo-

CANCELLARE UNA VOCE DI MENU

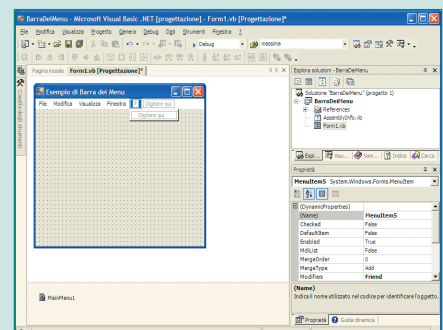
Per cancellare una voce di menu è sufficiente selezionare la voce di menu e premere il tasto *Canc*. Se la voce possiede dei sottomenu, verrà visualizzata una finestra di dialogo con il messaggio di conferma sull'operazione di cancellazione.

SOTTOMENU

Ogni volta che in fase di esecuzione compare una freccia al lato del menu, significa che tale voce possiede un sottomenu, questo effetto è gestito automaticamente da VB.Net

> IL MENU DESIGNER

4 Dopo aver trascinato il controllo sulla form, apparirà un nuovo elemento, denominato MainMenul, in un'area posizionata nella parte inferiore del form detta barra delle componenti. Viene mostrata, inoltre, la barra dei menu con il testo Digitare qui.

> INSERIAMO IL TITOLO

5 Nel punto in cui è visualizzato il testo Digitare qui, possiamo scrivere il titolo del menu, ad esempio: File. VB.NET visualizzerà il testo appena inserito come titolo e produrrà automaticamente una o più caselle Digitare qui per la creazione di nuovi titoli di menu e sottomenu.

> INSERIAMO LE VOCI

6 Allo stesso livello della voce File appena inserita, dobbiamo cliccare sulla casella Digitare qui ed inserire la nuova voce Modifica.
Con lo stesso Inseriamo i menu: *Visualizza, Finestra, ?* che rappresentano le voci di menu standard.

no invisibili tutti i suoi sottomenu. Nascondere voci di menu è un modo per controllare l'interfaccia utente e limitare il numero dei comandi accessibili. Anche in questo caso si possono rendere visibili o invisibili i menu da codice

```
MnuModifica.Visible = True
MnuModifica.Visible = False
```

Se nascondiamo un menù è assolutamente necessario anche disattivarlo, in quanto l'averlo nascosto non impedisce l'accesso ad un comando di menu tramite un tasto di scelta rapida.

NOTA

MODIFICARE IL MENU A RUNTIME

È possibile utilizzare la proprietà SourceControl per determinare l'ultimo controllo in cui è stato visualizzato il menu pop-up per eseguire operazioni specifiche del controllo o modificare il menu di scelta rapida visualizzato per il controllo. Se il menu di scelta rapida non è visualizzato in alcun controllo, la proprietà restituirà il valore riferimento Nothing. È possibile utilizzare questa proprietà nell'evento Popup per accertarsi che nel controllo siano visualizzate le voci di menu corrette.

I MENU STANDARD E L'EVENTO CLICK

È ormai quasi uno standard (nonché un consiglio Microsoft) la disposizione ed il contenuto di alcuni menu, brevemente in ordine da sinistra verso destra:

- **Il menu File** dovrebbe contenere i comandi necessari per le operazioni su file quali le funzioni: *Nuovo, Apri, Chiudi, Salva,* nonché la voce *Esci* per uscire dal programma.
- **Il menu Modifica** dovrebbe contenere i comandi correlati alla selezione e modifica di oggetti o testi quali: *Annulla, Taglia, Copia, Incolla.*
- **Il menu Visualizza** dovrebbe contenere i comandi che permettono di visualizzare o meno la barra degli strumenti o la barra di stato.
- **Il menu Finestra** dovrebbe contenere i comandi per la disposizione, l'apertura, la chiusura, ed il passaggio tra le diverse finestre aperte.
- **Il menu ? (Help)** dovrebbe contenere i comandi per ottenere un aiuto dal programma.

I menu peculiari di un'applicazione dovrebbero essere inseriti tra il menu *Visualizza* ed il menu *Finestra.* L'evento *Click* sarà certamente l'evento più utilizzato per la gestione dei menu e viene attivato, com'è facilmente intuibile, quando l'utente clicca con il mouse su una voce menu (oppure quando seleziona una voce di menu utilizzando i tasti di scelta rapida).
Ad esempio:

```
Private Sub MnuFileApri_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs)
                              Handles MnuFileApri.Click
  'Codice necessario per la gestione dell'apertura di
                                                un file
  '...........
  OpenFileDialog1.ShowDialog()
  '...........
End Sub
```

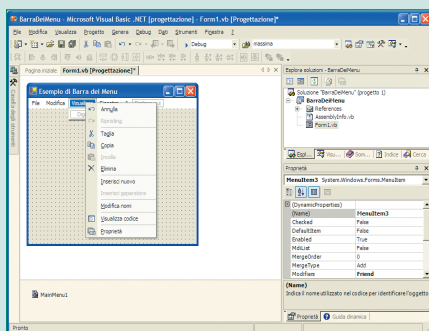
Per gestire le operazioni necessarie all'apertura di un file, si può usare il controllo *OpenFileDialog* che descriveremo nei prossimi articoli.

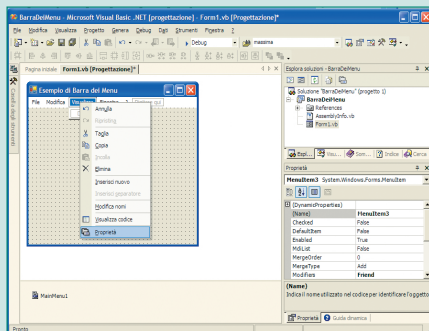
MENU CONTESTUALI (POP-UP)

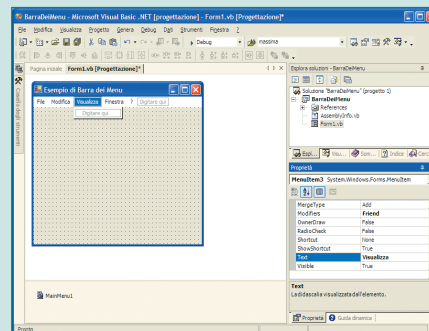
I *menu Pop-up* sono i menu, sensibili al contesto, che vengono visualizzati quando l'utente clicca con il tasto destro su un oggetto dell'interfaccia. La procedura per creare e mostrare menu di contesto è molto simile a quella appena descritta per i menu standard: è sufficiente trascinare un oggetto *ContextMenu* sulla form e impostare graficamente la struttura del menu come si è visto in precedenza.

MODIFICARE UNA VOCE DI MENU

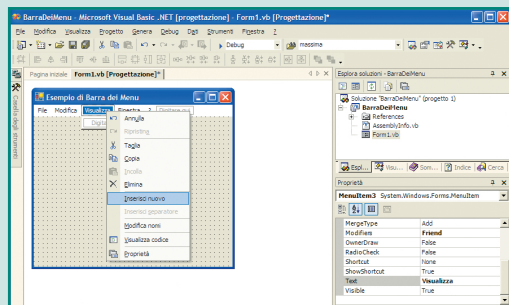
Un'operazione che si verificherà spesso sarà quella di aggiungere, rinominare, o eliminare voci di menu. Vediamo come

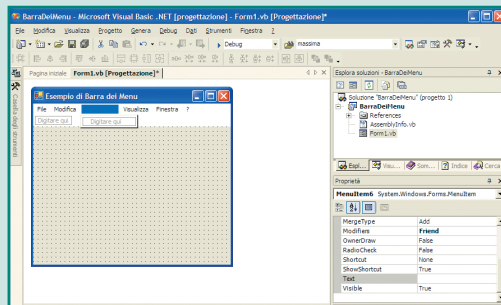
> AVVIARE IL DESIGNER

1 Per modificare una voce di menu, clicchiamo sulla barra dei menu, in modo da visualizzare il menu a discesa contestuale.

> SELEZIONARE LA VOCE

2 Dal menu a discesa, dobbiamo selezionare la voce proprietà in modo da visualizzare la finestra delle proprietà.

> APPORTARE LA MODIFICA

3 Infine, dobbiamo cliccare sulla casella a destra della proprietà Text e variare l'etichetta in quella desiderata.

INSERIRE UNA NUOVA VOCE DI MENU TRA QUELLE GIÀ INSERITE

> AVVIARE IL DESIGNER

1 Per inserire una nuova voce di menu in mezzo a due voci già inserite, possiamo cliccare con il tasto destro del mouse sulla barra dei menu in modo da visualizzare il menu a discesa.

> SELEZIONARE LA VOCE

2 Dal menu a discesa selezioniamo *Inserisci Nuovo,* In questo modo viene visualizzata una casella vuota a sinistra della voce selezionata, in cui possiamo inserire il valore dell'etichetta.

L'unica differenza è che tutte le voci di menu devono essere create a partire da un nodo fittizio denominato *ContextMenu*.

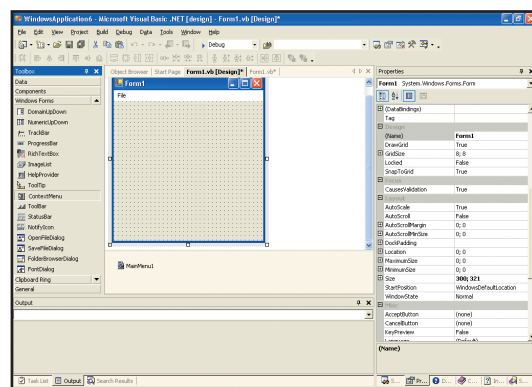



Fig. 1: ***Per realizzare un menu contestuale è necessario copiare un componente contextmenu sulla form ed editarlo***

Dopo aver disegnato la struttura del menu di contesto, è necessario associarlo ad uno o più controlli della form, oppure all'oggetto *Form* stesso, solo in questo modo il menu viene visualizzato. Tutti i controlli a cui è possibile associare un menu di contesto, espongono la proprietà *ContextMenu,* per questo è sufficiente cliccare sul pulsante con i tre puntini accanto alla proprietà, e selezionare il menu di contesto da associare al controllo.
È possibile associare un oggetto *ContextMenu* a più controlli (che, per questo, condivideranno lo stesso menu di contesto), oppure associare oggetti *ContextMenu* diversi a controlli differenti. Anche per i menu di contesto, il codice necessario a far compiere l'azione di competenza, deve essere scritto nell'evento *Click*. Un ulteriore evento messo a disposizione dall'oggetto *ContextMenu,* è l'evento *Popup* che viene generato nel momento in cui il menu diventa visibile. È possibile utilizzare questo evento, ad esempio, per mostrare ulteriori informazioni nella barra di stato, oppure per impostare segni di spunta, disattivare elementi ed eseguire altre operazioni di menu.

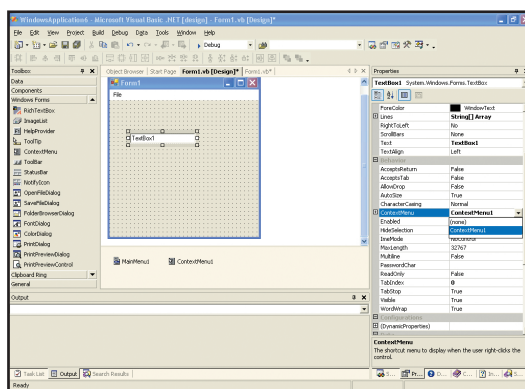



Fig. 2: ***Per associare un menu popup ad un oggetto è necessario utilizzare la proprietà context menu dell'oggetto ed associarla al menu precedentemente preparato***

CONCLUSIONI

In questo articolo ci siamo occupati di descrivere tutti gli strumenti necessari per progettare e disegnare un sistema di menu, nel prossimo articolo porteremo a termine la descrizione dei sistemi di menu, inquadrandoli in un'ottica più ampia: la scelta della tipologia d'interfaccia.

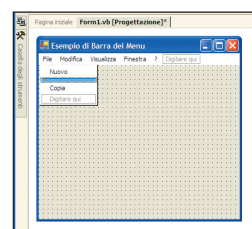
Luigi Buono

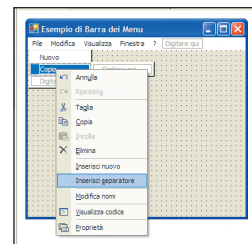
NOTA

INSERIRE LE BARRE DI SEPARAZIONE

Per inserire le barre di separazione di menu possiamo usare due metodi

TRAMITE LA PROPRIETÀ TEXT

Dalla finestra delle proprietà, possiamo selezionare la proprietà Text e porla pari a: - (trattino).

TRAMITE IL MENU DESIGNER

Possiamo cliccare con il pulsante destro del mouse nella posizione in cui si desidera inserire la barra, e dal menu a tendina selezionare il comando *Nuovo separatore.*

Aree Login e Password pronti

ASP.NET mette a disposizione dello sviluppatore un potente meccanismo per gestire l'autenticazione e l'autorizzazione per l'accesso alle risorse: FormsAuthentication. Ecco come funziona!

Il primo passo per l'attuazione di politiche di sicurezza su un sistema informatico è sempre costituito dall'implementazione di un sistema di gestione delle credenziali di autenticazione. In parole povere la buona vecchia accoppiata Login /Password rappresenta il primo step per concedere l'utilizzo di determinate risorse in un applicativo. In questo articolo ci occuperemo di capire come in ASP.NET possa essere gestito facilmente un meccanismo di autenticazione potente e funzionale. Allo scopo utilizzeremo una caratteristica di ASP.NET nota come *FormsAuthentication*

PERCHÉ FORMSAUTHENTICATION?

Letteralmente *FormsAuthentication* vuol dire "Autenticazione via form". In pratica si tratta di costruire una semplice applicazione che, sfruttando una form HTML, invii al server i dati di autenticazione ovvero username e password, in modo che quest'ultimo possa autenticare l'utente e garantire l'accesso sia solo a chi è in possesso della giuste credenziali.
Il primo passo sarà creare una pagina html di login, in modo che sia possibile collegarsi ad un database e confrontare i dati inseriti dall'utente con quelli presenti nel DB. Poiché il web è *state-less*, per tenere traccia dello stato dell'autenticazione ASP.NET utilizza un semplice cookie. Le informazioni contenute all'interno di questo cookie possono essere criptate, attraverso un'opzione, in modo che non sia possibile risalirne al contenuto. Ovviamente un sistema del genere non funzionerebbe nel caso in cui il browser non supporti i cookie, ma la stessa identica limitazione la avremmo utilizzando un oggetto *Session*.
In applicazioni Classic ASP (o con tecnologie equivalenti in quanto a funzionalità, come PHP) tipicamente lo stato è salvato in un contenitore quale la *Session*. Salvare lo stato direttamente in un cookie ha il vantaggio che è possibile definirne scadenze personalizzate, ad esempio per fare in modo che l'utente possa essere autenticato per un tempo piuttosto ampio.

L'AUTENTICAZIONE NON È L'AUTORIZZAZIONE

Con ASP.NET il concetto di autenticazione è profondamente slegato da quello di autorizzazione. Per costruire applicazioni che abbiano aree protette è necessario comprendere a fondo entrambi i concetti, dato che sono strettamente legati tra di loro. Nel primo caso ricade la parte prettamente dedicata al riconoscimento dell'utente. Nel secondo caso ASP .NET dovrà consentire o meno l'accesso ad una risorsa utilizzando le credenziali fornite in fase di autenticazione e confrontandole con una lista di permessi. Gli utenti autenticati potranno accedere a quella parte di risorsa che i permessi ad essa associati gli concedono. Ad esempio un certo utente avrà il permesso di accedere ad una parte del disco ma non ad un'altra. La parte relativa alla verifica delle credenziali di autenticazione ricade propriamente sotto le responsabilità della *FormsAuthencation*.
La parte relativa ai permessi viene invece gestita in automatico da ASP.NET.

Struttura tabella 'users' in 'authentication' in '(local)'

Nome colonna	Tipo di dati	unghezza	Ammetti Null
ID	int	4	
username	varchar	50	
password	varchar	50	

Fig. 1: **Struttura della tabella Utenti**

PRIMA DI COMINCIARE: LA CONFIGURAZIONE

Il sistema utilizzato in applicazioni ASP.NET per la configurazione centralizzata dell'applicazione è un semplice file chiamato *web.config*. Se non ne abbiamo già uno, dobbiamo creare un file di nome *web.config* nella root della nostra applicazione, ed

aggiungere questo semplice codice:

```
<configuration>
<system.web>
<authentication mode="Forms">
<forms
name="ioProgrammo"
path="/"
loginurl="login.aspx"
protection="All|None|Encryption|Validation"
timeout="10"
slidingExpiration="true"
/>
</authentication>
</system.web>
</configuration>
```

In questo modo ogni richiesta alla nostra applicazione verrà rediretta ad una pagina chiamata *login .aspx*. Con gli attributi *name* e *path* impostiamo rispettivamente il nome del cookie in cui salveremo lo stato dell'autenticazione ed il suo percorso di validità. Con *timeout* invece impostiamo la validità del cookie, in minuti, mentre *slidingExpiration* fa sì che, ad ogni richiesta, il timeout dell'autenticazione venga resettato, allungandone di fatto la scadenza.
L'attributo protection, per finire, specifica quale livello di protezione si vuole garantire al nostro cookie. Il migliore è *All*, che combina sia *Encryption*, che cripta le informazioni, che *Validation*, che effettua un controllo su quanto contenuto nel cookie. Quello da evitare è ovviamente *None*, che lascia il cookie in chiaro. Una volta definito questo primo passo, la parte di autenticazione da configurare in maniera centralizzata termina qui. Non ci resta che dare un'occhiata a come costruire la pagina di autenticazione.

Struttura tabella 'roles' in 'authentication' in '(local)'

	Nome colonna	Tipo di dati	Lunghezza	Ammetti Null
▶🔑	ID	int	4	
	role	varchar	50	
	username	varchar	50	

Fig. 2: ***Struttura della tabella per l'autenticazione***

AUTENTICARE L'UTENTE

A questo punto non ci resta che costruire la pagina di login. Partiremo con la costruzione di un database contenente due tabelle che ci consentano di salvare gli utenti e di stabilire i ruoli a cui un certo utente può avere accesso. La prima tabella, di nome *users*, conterrà due campi in cui salvare username e password, la seconda tabella avrà come nome *roles* e conterrà due campi, uno per il ruolo e l'altro per lo username associato. Creeremo anche una relazione tra le due tabelle, sui campi *username*. In tutti gli esempi andremo ad usare come classi per l'accesso ai dati quelle di *Ole-db*, in modo che sia possibile riadattare l'esempio anche ad un utilizzo con Access, nonostante la nostra applicazione in realtà faccia utilizzo di SQL Server. Una volta costruite le tabelle, dobbiamo procedere alla creazione della maschera di input all'interno del file *login.aspx*. Sarà sufficiente aggiungere una webform con due campi, uno per lo username e l'altro per la password, un pulsante a cui andremo ad associare un evento ed una label per mostrare eventualmente all'utente un messaggio di errore qualora l'autenticazione non andasse a buon fine. A questo punto dobbiamo personalizzare l'evento *DoLogin*, che abbiamo associato alla pressione del pulsante, cominciando a sfruttare uno dei metodi della classe *FormsAuthentication*, ovvero *RedirectFromLoginPage*. Prima di ogni cosa dovremo effettuare la query al database, avendo cura di fare il replace dell'apice in modo da evitare attacchi di SQL *Injection* (in questo caso, in produzione è sempre meglio utilizzare query parametriche, anche con Ole-db). Verificato che l'utente effettivamente esiste, andremo a richiamare il metodo statico *RedirectFromLoginPage*, passandogli come parametri lo username dell'utente ed un boolean che indica se il cookie è persistente o meno. In questo caso lo abbiamo impostato a *false*, specificando *true*, invece il cookie non verrebbe cancellato alla chiusura del browser, ma rimarrebbe attivo finché fino al raggiungimento del valore di timeout impostato in *web.config*. L'evento *login* sarà così gestito:

```
Sub DoLogin(objSender As Object, objArgs As EventArgs)
  Dim stringadiconn as String = "personalizzata"
  ' apertura connessione
  Dim conn as new OleDbConnection(stringadiconn)
  conn.Open
  ' stringa SQL per estrarre utente
  Dim SQL as String = "SELECT username FROM
       users WHERE username='" & txtUsername.Value
              .Replace("'", "''") & "' AND password='" &
               txtPassword.Value.Replace("'", "''") & "'"
  Dim command as new OleDbCommand(SQL, conn)
  Dim dr as OleDbDataReader = command.ExecuteReader()
  ' verifico che ci sia un utente
  Dim logged as Boolean = false
  If dr.Read() Then
    Logged = true
  Else
    status.Text = "<b>Username o password errati</b>"
  End If
  ' chiusura fisica di datareader e connessione
  dr.Close()
  conn.Close()
  if logged then
    FormsAuthentication.RedirectFromLoginPage(
                                   txtUsername.Value, False)
  end if
End Sub
```

AUTORIZZAZIONE CON ASP.NET 2.0?

La *FormsAuthentication* di ASP.NET 1.x non è proprio banale e specie per l'integrazione con i ruoli necessita di un po' di lavoro da parte dello sviluppatore. La versione 2.0 di ASP.NET, in uscita per novembre 2005, rende queste operazioni molto più semplici grazie all'utilizzo di nuove funzionalità che prendono il nome di Membership APIs. Quello che non tutti sanno è che ne è stato fatto un back porting per la versione 1.1 di ASP.NET, che è disponibile gratuitamente dall'URL http://download.microsoft.com/download/3/2/E/32E73F6A-2EF9-4F9D-AD04C7952789FF26/MSDNProviderModel.msi **ed è compatibile con quella che sarà distribuita con la 2.0, rendendo tra l'altro la migrazione anche più semplice.**

ED I RUOLI?

Come si può notare, finora non abbiamo ancora minimamente parlato di ruoli, perché in questo caso il discorso si complica. La versione 1.x di ASP.NET non ha infatti un supporto semplice alla creazione di autenticazione con ruoli ed è quindi necessario armarsi di un po' di pazienza per aggiungere questa funzionalità. Per prima cosa è necessario riscrivere la pagina di login, in modo che sia possibile estrarre i ruoli associati all'utente dalla tabella *roles*. Per fare questo, in prossimità della riga che richiama il metodo *RedirectFromLoginPage* andremo ad aggiungere una chiamata alla nostra funzione *Authenticate*, che si occuperà anche di estrarre i ruoli.
Il metodo *RedirectFromLoginPage*, in realtà richiama alcuni metodi della classe *FormsAuthetication*, costruendo quello che viene comunemente chiamato ticket di autenticazione, che altro non è che un contenitore che specifica i dati di autenticazione (ed

Fig. 3: **La pagina di Login**

i relativi ruoli) da associare all'utente, sfruttando una classe che è appunto chiamata *FormsAuthenticationTicket*. Nella nostra funzione di autenticazione dovremo quindi creare un ticket, passandogli le informazioni sullo username, sulla data di scadenza e sui ruoli, che andremo da estrarre con la funzione *GetRoles()*.
Fatto questo dovremo creare manualmente il cookie di autenticazione, sfruttando la classe *HttpCookie*, ed aggiungerlo alle header di risposta. Infine, andremo a leggere il valore del parametro *ReturnUrl* che ASP.NET utilizza per far tornare l'utente alla pagina su cui è arrivato prima di effettuare il login. Il tutto, tradotto in codice, diventa:

```
Sub Authenticate(Username as String)
  ' preparo l'autenticazione
  FormsAuthentication.Initialize()
  Dim roles as String = GetRoles(Username)
  ' genero il ticket
  Dim fat as FormsAuthenticationTicket = new
              FormsAuthenticationTicket(1, Username,
        DateTime.Now, DateTime.Now.AddMinutes(20),
     false, roles, FormsAuthentication.FormsCookiePath)
  ' creo il cookie di autenticazione
  Dim cookie as HttpCookie = new HttpCookie(
              FormsAuthentication.FormsCookieName,
                     FormsAuthentication.Encrypt(fat))
  Response.Cookies.Add(cookie)
  ' Return URL dopo autenticazione
  if Request("ReturnUrl") is Nothing then
    Response.Redirect("/")
  end if
  Response.Redirect(Request("ReturnUrl"))
End Sub
```

La funzione *GetRoles* è semplice, in quanto non fa altro che estrarre, dato lo username, tutti i ruoli a cui appartiene, salvandoli in una stringa, separati da ";":

```
' estrazione dei roles dal database, separati da ;
Function GetRoles(Username as String) as String
  ' apertura connessione
  Dim conn as new OleDbConnection(stringadiconn)
  conn.Open
  ' stringa SQL per estrarre utente
  Dim SQL as String = "SELECT role FROM roles
   WHERE username='" & Username.Replace("'", "''")
                                              & "'"
  Dim command as new OleDbCommand(SQL, conn)
  Dim dr as OleDbDataReader =
                           command.ExecuteReader()
  Dim roles as new StringBuilder()
  ' verifico che ci sia un utente
  While dr.Read()
    roles.Append(dr("role").ToString())
    roles.Append(";")
  End While
  ' chiusura fisica connessione
  dr.Close()
  conn.Close()
  ' restituisco i ruoli estratti dal database
  return roles.ToString()
End Function
```

ASSEGNARE I RUOLI ALL'UTENTE

A questo punto il ticket conterrà sia lo username, come nel caso del metodo *RedirectFormLoginPage*, sia i ruoli, cosa che con questo metodo non è possibile specificare. Tuttavia ancora non c'è un legame stretto fra i ruoli e l'applicazione. Di fatto i ruoli sono associati all'utente ma non sappiamo come concedere o meno l'accesso alle funzionalità sulla base di quanto fatto fino a questo momento.
In realtà ASP.NET utilizza il concetto di *Principal*, che si traduce in un insieme di classi che fanno sì che l'utente venga rappresentato attraverso specifiche funzionalità. Dovremo dunque riscrivere il *Principal* associato alla richiesta corrente, che ASP.NET espone attraverso la proprietà *User* di *HttpContext*, che è una classe che rappresenta il contesto corrente di esecuzione. Per far sì che ASP .NET riconosca il nostro utente, dovremo quindi andare ad intercettare l'evento *AuthenticateRequest*, che si verifica quan-

NOTA

DELLE CREDENZIALI

Se non volete demandare l'autorizzazione alle funzionalità di ASP.NET, attraverso il *web.config*, è sempre possibile effettuare in maniera programmatica questi controlli, sfruttando i metodi dell'interfaccia IPrincipal, di cui GenericPrincipal è un'implementazione. Per controllare che l'utente sia autenticato:

```
User.IsAuthenticated
```

Mentre per verificare che appartenga al ruolo admin:

```
User.IsInRole("admin")
```

do una richiesta viene autenticata. Per farlo abbiamo principalmente due strade: creare un *HttpModule* o usare il *global.asax*.
Nel *global.asax* andremo dunque a leggere dal ticket di autenticazione, che è salvato in maniera criptata nel cookie, i ruoli che abbiamo precedentemente salvato, ricavandoli dal database. Dato che i ruoli devono essere specificati come un array di stringhe, useremo il metodo *Split* della classe *String* per dividere i nostri ruoli, separati dal carattere ";", in un array. Per finire dovremo riscrivere il *Principal* associato alla proprietà *User* di *HttpContext* creandone uno nuovo, con *GenericaPrincipal*, che nel costruttore accetta una classe che implementa l'interfaccia *IIdentity*, come *FormsIdentity* che è specifica della *Forms Authentication*, ed appunto un array di stringhe con i ruoli.

```
<%@ Import namespace="System.Security"%>
<%@ Import namespace=
                    "System.Security.Principal"%>
<SCRIPT RUNAT="SERVER" LANGUAGE="VB">
' l'utente cerca di autenticarsi
Sub Application_AuthenticateRequest(ByVal sender
                  As Object, ByVal e As EventArgs)
  ' carico l'utente e recupero il ticket
  if Not HttpContext.Current.User is Nothing then
    Dim identity as FormsIdentity = CType(
       HttpContext.Current.User.Identity, FormsIdentity)
    Dim ticket as FormsAuthenticationTicket =
                                     identity.Ticket
    ' carico i roles dal ticket del cookie
    Dim roles as String() = ticket.UserData.Split(";")
    ' sovrascrivo User con un nuovo GenericPrincipal
    HttpContext.Current.User = new
                     GenericPrincipal(identity, roles)
  end if
End Sub
</SCRIPT>
```

Fatto questo la richiesta corrente sarà autenticata ed i ruoli assegnati direttamente all'utente, in modo che si possano sfruttare i meccanismi di autorizzazione di ASP.NET in maniera nativa, senza fare altro.

AUTORIZZAZIONE DA WEB.CONFIG

L'autenticazione, come visto, è solo la prima parte da implementare per la protezione di un'applicazione ASP.NET. Fatta l'autenticazione, dobbiamo impostare i permessi di accesso alle risorse, specificando le nostre *policy* di autorizzazione. Anche in questo caso è necessario modificare il *web.config*, che essendo il sistema demandato alle configurazioni dell'applicazione, andrà a contenere le nostre *policy* di accesso alle risorse.

Per capire quanto sia semplice specificare le *policy* di accesso, supponiamo di voler garantire ad una pagina che, con molta fantasia, chiameremo *pagina.aspx*, l'accesso solo agli utenti autenticati.
Apriamo il nostro *web.config* e prima dell'ultimo tag (che è *</configuration>*) inseriamo:

```
<location path="pagina.aspx">
<system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
```

L'effetto è che la pagina con il nome specificato non potrà essere visitata (chiave *deny*) dagli utenti *?*, che corrispondono a tutti quelli che non hanno fatto l'autenticazione. Se invece volessimo garantire l'accesso a tutti gli utenti autenticati è sufficiente specificare come valore ***.
È poi possibile anche specificare una lista di utenti da bloccare, inserendo lo *username*. Nel caso in cui invece si volesse specificare una lista di utenti a cui è consentito l'accesso, è necessario sostituire *deny* con *allow*. Per quanto riguarda i ruoli il discorso è esattamente lo stesso, come si può vedere nel seguente esempio in cui il ruolo *admin* ha accesso e quello *user* è invece bloccato

```
<location path="admin.aspx">
<system.web>
    <authorization>
      <deny roles="user" />
      <allow roles="admin" />
    </authorization>
  </system.web>
</location>
```

Ovviamente è possibile combinare sia politiche di accesso per ruolo che per utente. Infine, l'attributo *path* del tag *location* rappresenta il percorso relativo, primo / escluso, rispetto alla root dell'applicazione in cui gira l'autenticazione. Si può anche specificare una directory ed in questo modo la protezione è estesa a tutte le risorse ASP.NET che contiene.

CONCLUSIONI

Proteggere l'accesso a parti riservata, anche con i ruoli, non è difficilissimo. ASP.NET ci mette a disposizione, con la *FormsAuthentication* ed i meccanismi di autorizzazione, un sistema su cui costruire facilmente le nostre personalizzazioni. E con la prossima versione di ASP.NET, la 2.0, sarà ancora più semplice.

Daniele Bochicchio

L'AUTORE

Daniele Bochicchio è il content manager di *ASPItalia.com*, community che si occupa di ASP.NET, Classic ASP e Windows Server System. Il suo lavoro è principalmente di consulenza e formazione, specie su ASP.NET, e scrive per diverse riviste e siti. È Microsoft ASP.NET MVP, un riconoscimento per il suo impegno a supporto delle community e per l'esperienza maturata negli anni. Il suo blog è all'indirizzo http://blogs.aspitalia.com/daniele/

Matematica mon amour...!

Spesso durante lo sviluppo di una web application ci siamo trovati a fare i conti con numeri e formule matematiche. JavaScript offre una nutrita scelta di funzioni per risolvere questo problema

Ok, l'idea è molto semplice e l'avrete già vista realizzata su Internet più di una volta. Implementeremo una calcolatrice scientifica con JavaScript. Ovviamente non è il valore commerciale dell'idea e della seguente applicazione, quello da considerare in questo tipo di articolo. Piuttosto perseguiremo un fine didattico addentrandoci nelle classi JavaScript indispensabili per scrivere applicazioni matematiche con JavaScript. In particolare ci occuperemo della classe *Math* utilizzata per la gestione delle operazioni e funzioni matematiche e della classe *Number* utilizzata per la manipolazione delle entità numeriche.

REQUISITI

Conoscenze richieste: Basi di Javascript

Software: Notepad

Impegno

Tempo di realizzazione

CLASSE MATH

Contiene l'insieme delle funzioni matematiche che possono essere utilizzate durante i calcoli scientifici. In particolare l'elenco delle costanti supportate è riportato in **Tabella 1.** Mentre l'elenco delle funzioni supportate è riportato in **Tabella 2.** Supponiamo ad esempio di volere esprimere un angolo in radianti, e di doverlo convertire dai gradi sessagesimali, la formula matematica è nota ed è la seguente

$\pi : 180 = x\text{^} : x°$

dove si è posto:

a) $x\text{^}$ = *valore dell'angolo espresso in radianti*
b) $x°$ = *valore dell'angolo espresso in gradi*

con la formula inversa si ha:

$x\text{^}=(\pi * x°)/180$

La relazione può essere implementata nel seguente codice javascript:

```
function GradToRad(Gradi)
{ Rad = Gradi*Math.PI/180;
  return Rad; }
```

Un codice JavaScript che ritorna un valore generico espresso da una formula è il seguente:

```
function Calcola(Formula)
{  return eval(Formula); }
```

Per stampare il risultato, ad esempio, del seno di 90°, si può scrivere:

```
document.write( Calcola("Math.sin(GradToRad(90));"));
```

Nei paragrafi successivi presenteremo una applicazione pratica di questo spezzone di codice, che permetterà di calcolare espressioni complicate a piacere, utilizzando un minimo di interfaccia utente.

#	Costante	Descrizione	Valore approssimativo
1	*E*	Costante di Nepero, base dei logaritmi naturali.	2.718281…
2	*LN10*	Logaritmo naturale di 10.	2.3025…
3	*LN2*	Logaritmo naturale di 2	0.6931…
4	*LOG2E*	Il logaritmo in base due del numero E.	1.4426
5	*LOG10E*	Il logaritmo in base 10 di E.	0.4342…
6	*PI*	La costante pi greca.	3.1415..
7	*SQRT1_2*	La radice quadrata di _ .	0.7071…
8	*SQRT2*	La radice quadrata di 2.	1.4142

TABELLA1: L'elenco delle costanti supportate dalla classe Math

CLASSE NUMBER

La classe Number permette di gestire il modo in cui è presentato il formato di un numero; inoltre definisce i valori di alcune costanti limiti e che possono essere utilizzate nei calcoli. Un elenco delle costanti supportate dalla classe *Number* è riportato nella **Tabella 3**. Si noti che *NaN* non è lo stesso dell'equivalente omologo della classe *Global,* anche se poi in effetti coincidono esattamente come valore.

Un oggetto di tipo *Number* si instanzia nel solito modo:

```
var MioNumero = new Number(123.876);
```

Per stampare il valore del numero *MioNumero*, si ricorre al metodo *valueOf* della classe *Number*:. La stringa di codice:

```
document.write(myNumero.valueOf);
```

restituisce 123.876. Gli altri metodi che vengono utilizzati per formattare correttamente un numero sono i seguenti:

a) **Metodo toFixed(digits):** Restituisce una stringa rappresentante il numero in notazione *fixed-point*, con <*digits*> numeri dopo la virgola. Ad esempio:

```
var Numero = new Number(123.876);
document.write("var Numero = new Number(
                                  123.876)<br>");
document.write (Numero.toFixed(2));
```

restituisce: *123.88*

b) **Metodo toExponential(digits):** Restituisce una stringa rappresentante il numero in notazione esponenziale con una cifra prima del decimale significativo e <digits> cifre dopo il punto decimale. Ad esempio:

```
var Numero = new Number(123.876);
document.write("var Numero = new Number(
                                  123.876)<br>");
document.write (Numero.toExponential(2));
```

restituisce: *1.24e+2*

c) **Metodo toPrecision(precisione):** Restituisce un stringa rappresentante in numero con:

   a. Una cifra prima del punto decimale e <precisione> -1 cifre dopo il punto decimale, nel caso di notazione esponenziale
   b. Una cifra prima del punto decimale e <precisione> cifre dopo il punto decimale, nel caso di notazione fixed-point. Ad esempio:

```
var Numero = new Number(123.8764567);
document.write("var Numero = new Number(
                              123.8764567)<br>");
document.write (Numero.toPrecision());
document.write (Numero.toPrecision(2));
document.write (Numero.toPrecision(3));
document.write (Numero.toPrecision(4));
document.write (Numero.toPrecision(5));
```

#	Funzione	Descrizione
1	*abs(x)*	Restituisce il valore assoluto del numero *x*
2	*acos(x)*	Restituisce l'arcocoseno di *x*, espresso in radianti e compreso tra *+0* e π
3	*asin(x)*	Restituisce l'arcoseno di *x*, espresso in radianti e compreso tra -π/2 e +π/2
4	*atan(x)*	Restituisce l'arcotangente di *x*, espresso in radianti e compreso tra -π/2 e +π/2
5	*atan2(y,x)*	Restituisce il valore dell'arcotangente del quoziente *y/x*, usando il segno di *y* ed *x* per stabilire in quale quadrante inserire il risultato, che è espresso in radianti ed è compreso tra -π e +π
6	*ceil(x)*	Restituisce il più piccolo intero che più si avvicina ad *x*, rimanendo maggiore di *x*. Se *x* è intero, il risultato è sempre *x*.
7	*cos(x)*	Restituisce il coseno del numero *x*, espresso in radianti.
8	*exp(x)*	Restituisce l'esponenziale del numero *x*, ossia la costante di nepero *E* elevata al numero *x*.
9	*floor(x)*	Restituisce il più piccolo intero che più si avvicina ad x, rimanendo minore di *x*. Se *x* è intero, il risultato è sempre *x*.
10	*log(x)*	Restituisce il logaritmo naturale del numero *x*
11	*max(x1,x2,.,xn)*	Restituisce il valore massimo tra i numeri *x1, x2, ..,xn.*
12	*min(x1,x2,…,xn)*	Restituisce il valore minimo tra i numeri *x1, x2, ..,xn.*
13	*pow(x,y)*	Restituisce il valore di *x* elevato alla potenza *y, xy*
14	*random()*	Restituisce un numero generato in modo semicasuale maggiore di zero e minore di uno, con distribuzione quasi uniforme. Non accetta argomenti in input.
15	*round(x)*	Ritorna il valore di *x* arrotondato ad un valore intero
16	*sin(x)*	Ritorna il valore del seno di *x*, essendo *x* espresso in radianti.
17	*sqrt(x)*	Restituisce la radice quadrata di *x*, se *x* è non negativo.
18	*tan(x)*	Restituisce la tangente di *x*, espresso in radianti.

TABELLA 2: l'elenco delle funzioni supportate dalla classe Math.

#	Costante	Valore	Descrizione
1	*Number.MAX_VALUE*	1.7976931348623157e+308	È il massimo valore numerico positivo gestibile
2	*Number.MIN_VALUE*	5e-324	È il minimo valore numerico positivo gestibile
3	*Number.NEGATIVE_INFINITY*	-Infinity	Equivale a – infinito
4	*Number.POSITIVE_INFINITY*	+Infinity	Equivale a +infinito
5	*Number.NaN*	NaN	Rappresenta un valore non numerico

TABELLA 3, le costanti supportate dalla classe Number

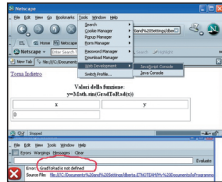
restituiscono rispettivamente: *123.8764567, 1.2* e +2, *124, 123.9, 123.88.*

UN CALCOLATORE PER LE FUNZIONI MATEMATICHE...

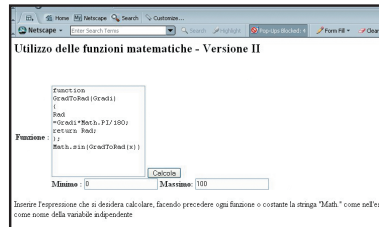
Come esempio di utilizzo delle classi discusse, realizziamo un'applicazione che è in grado di calcolare il valore di una funzione, complessa a piacere. In particolare, dobbiamo realizzare un'applicazione che è in grado di:

a) Calcolare il valore di una funzione in un punto qualunque del dominio di validità della variabile di input.
b) Calcolare una tabella di valori della funzione all'interno di un range di valori valido per il dominio.

INCONGRUENZE CON NETSCAPE

Lo script di calcolo della tabella dei valori della funzione si blocca se lanciato su Netscape 7.2. Utilizzando la console javascript di Netscape (accessibile dalla voce di menu: *Tool -> Web Development ->Javascript Console* si nota l'errore: *"La funzione GradToRad(..) non è definita"*,

In effetti la funzione *GradToRad()* è definita. Allora dove è l'errore? Iniziamo a vedere prima quale è la soluzione. Nella casella di testo *txtInput* inseriamo, oltre alla funzione che vogliamo calcolare anche la funzione *GradToRad()*. In pratica il contenuto della textarea sia il seguente:

```
function GradToRad(Gradi)
{ Rad = Gradi*Math.PI/180;
  return Rad;};
Math.sin(GradToRad(x))
```

Premendo il pulsante di calcolo, funziona tutto alla perfezione. Inoltre, inserendo lo stesso contenuto della textarea in Internet Explorer, anche in questo caso continua a funzionare. Quando si esegue la

routine *Calcola(...)* viene creata una nuova istanza della pagina, che diventa quella di default, nella quale viene creata la tabella. Andando a visualizzare il sorgente della tabella, si potrà vedere che in questa pagina NON È presente la routine *GradToRad()*, che quindi non risulta defnita. Internet Explorer, che ha una gestione delle istanze delle pagine differente rispetto a quella di Netscape, vede la pagina di partenza (quella con l'interfaccia utente) e quella di arrivo (quella con la sola tabella dei risultati) come la "stessa" istanza, mentre Netscape le vede come due istanze differenti. Questione di gusti.... Per risolvere alla radice il problema, occorre fare in modo che la tabella venga creata in una nuova finestra (ad esempio un popup) nella quale si inserisce la routine *GradToRad().;* i popup verranno analizzati in articoli successivi.

Iniziamo con il primo punto. L'interfaccia della nostra applicazione è quella riportata in **Figura 1**, nella quale i campi in gioco sono:

#	Nome campo	Descrizione
1	*frmCalcola*	Form contenitore delle caselle di testo e pulsante di interfaccia
1	*txtInput*	Campo di tipo TextArea contenente la formula da calcolare
2	*txtOutput*	Campo di testo contenente il valore del risultato calcolato
3	*btnEsegui*	Pulsante che lancia lo script Calcola(<valore di input>)

Fig. 1: L'interfaccia dell'applicazione

Il punto centrale della nostra applicazione è lo script *Calcola(<Formula in input>)* che accetta in input il valore che è stato inserito all'interno della casella di testo *txtInput*. Il codice dello script è il seguente:

```
function Calcola(Formula)
{  if(Formula != "")
     document.frmCalcola.txtOutput.value =
                                    eval(Formula);
}
```

Ad esempio, una formula che si potrebbe volere calcolare è: *Math.sin(GradToRad(90))*. Si noti che è stata utilizzata la funzione *GradToRad()* precedentemente definita ed il valore che si passa è pari a 90° sessagesimali. Il codice HTML che crea l'interfaccia mostrata in **Figura 1** e che richiama la routine calcola è il seguente:

```
<form name="frmCalcola" id="frmCalcola">
<table>
  <tr>
  <td><b> Funzione : </b></td>
  <td>
  <textarea name="txtInput" rows="10"
     cols="20">Math.sin(GradToRad(90))</textarea>
  <input type="button" name="btnEsegui"
               value="Calcola" onClick="Calcola(
         document.frmCalcola.txtInput.value)">
  </td>
  </tr>
  <tr>
  <td><b>Risultato: </b></td>
  <td><input type="text" name="txtOutput"
                     id="txtOutput" value=""></td>
  </tr>
</table>
</form>
```

La parte importante è quella mostrata in grassetto che richiama, alla pressione del tasto *btnEsegui*, la routine calcola, passandogli il contenuto della *textarea* denominata *txtInput*. La routine calcola, tramite la funzione *eval()* effettua l'elaborazione della formula che riceve in input (e che si chiama proprio *Formula*) e con questa valorizza direttamente il valore della casella di output *txtOutput*. Veniamo al secondo punto. Quello che vogliamo fare è generalizzare la prima applicazione, facendo in modo di stampare non solo più un valore puntuale della funzione, ma una tabella di valori. L'interfaccia della nostra nuova applicazione è la seguente:

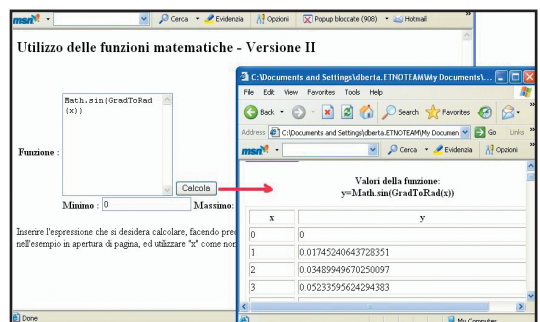



Fig. 2: L'interfaccia dell'applicazione generalizzata

I campi di interesse sono:

#	Nome campo	Descrizione
1	*frmCalcola*	Form contenitore delle caselle di testo e pulsante di interfaccia
1	*txtInput*	Campo di tipo *TextArea* contenente la formula da calcolare
2	*txtMin*	Campo di testo contenente il valore minimo della variabile indipendente *x*
	txtMax	Campo di testo contenente il valore massimo della variabile indipendente *x*
3	*btnEsegui*	Pulsante che lancia lo script *Calcola(<Funzione>,<Min>,<Max>)*

Lo script *Calcola(…),* è un po' più complicato; adesso accetta in input tre valori, che sono: la Funzione che deve calcolare, il valore minimo della variabile *x* da cui partire con il calcolo e il valore massimo della stessa. Inoltre, la funzione deve essere espressa in termini di variabile *x*, e non più inserendo un valore fisso, come nel caso precedente. Come prima, la formula che si potrebbe volere calcolare è: *Math.sin (GradToRad(x))*. È sempre stata utilizzata la funzione *GradToRad()* precedentemente definita ma il valore che gli si passa NON È un valore fisso, ma la variabile indipendente *x*. La routine calcola è ora la seguente:

```
function Calcola(Formula,Min,Max)
{ document.write("<p><a href='#' onClick=
                    'javascript:history.back()'>Torna
                              Indietro</a></p>\n");
  document.write("<table border='1' width='500'
                    celpadding='1' cellspacing='5'>\n");
  document.write("<caption><b>" + "Valori della
                    funzione: <br>" + "y=" + Formula +
                              "<b></caption>\n");
  document.write("<thead>\n");
  document.write("<tr>\n");
  document.write("<th>\n");
  document.write("<b>x</b>\n");
  document.write("</th>\n");
  document.write("<th>\n");
  document.write("<b>y</b>\n");
  document.write("</th>\n");
  document.write("</thead>\n");
  document.write("<tbody>\n");
  for(var x=Min;x<=Max;x++)
  {  document.write("<tr>\n");
     document.write("<td>\n");
     document.write(x);
     document.write("</td>\n");
     document.write("<td>\n");
     document.write(eval(Formula));
     document.write("</td>\n");
  }
  document.write("</tbody>\n");
  document.write("</table>");
}
```

La maggior parte del codice serve per costruire la tabella. Il "cuore" della routine è il ciclo in grassetto, che permette il calcolo ripetuto del valore della funzione per i differenti valori della variabile x contenuti nell'intervallo *[Min,Max]* considerato.
Il codice HTML che serve per richiamare la routine *Calcola(..)* è il seguente:

```
<form name="frmCalcola" id="frmCalcola">
<table>
  <tr>
    <td><b> Funzione : </b></td>
    <td>
      <textarea name="txtInput" rows="10" cols=
            "20">Math.sin(GradToRad(x))</textarea>
      <input type="button" name="btnEsegui"
         target= "new" value="Calcola" onClick="
      Calcola(document.frmCalcola.txtInput.value,
      document.frmCalcola.txtMin.value,document
                     .frmCalcola.txtMax.value)">
      <br>
      <b>Minimo :</b> <input type="text" name=
                  "txtMin" id="txtMin" value="0">
      <b>Massimo:</b> <input type="text" name=
               "txtMax" id="txtMax" value="100">
    </td>
  </tr>
</table>
</form>
```

Nella parte in grassetto, alla pressione del tasto btnEsegui viene passato alla routine *Calcola(..)* il valore della funzione contenuta nella *TextArea*, il valore di minimo della variabile *x* contenuto nella casella di testo *txtMin* e quello di massimo contenuto nella casella *txtMax*.

CONCLUSIONI

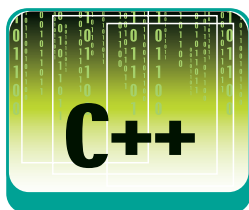
Nell'articolo abbiamo discusso dei metodi che servono per la gestione di oggetti *Math* e *Number*, presentando un esempio di applicazione utile al calcolo delle funzioni. L'applicazione presentata mostra evidenti limiti. In particolare:

i) Non ci sono controlli formali sulla validità dei campi inseriti
ii) Si è obbligati a far precedere ogni funzione dal prefisso *"Math."*
iii) Non viene gestita la parte di formattazione dei numeri (usando ad esempio i metodi della classe *Number*) ed il layout non è sicuramente dei migliori..

Se l'utente ha tempo e pazienza, può implementare tutte queste migliorie a scopo didattico…

Danilo Berta

Bluetooth e tutto comunica!

Realizziamo un'applicazione che trasferisce la rubrica dei contatti del telefono ad un computer standalone. Estendiamo questa funzionalità consentendo di comporre un numero dal PC

Nei precedenti articoli sono stati trattati quegli argomenti che ci hanno permesso di avere un quadro generale ben strutturato sull'architettura Symbian. Sfruttando i meccanismi di accesso alla rubrica e lo stack bluetooth costruiamo adesso qualcosa che potrebbe risultare utile nella vita di tutti i giorni. Cercheremo di trasferire, via etere, tutti i contatti dal nostro telefonino al PC di casa che, una volta ricevuti, si preoccuperà di visualizzarli in una graziosa GUI. Inoltre, per conferire un pizzico di interazione, faremo partire una telefonata semplicemente cliccando con il mouse su uno dei nostri contatti

PROGETTIAMO IL SISTEMA

Intuitivamente pensiamo subito di dover scrivere due software, uno per cellulare ed uno per pc. Essi comunicheranno tra loro usando il classico modello Client-Server, molto conosciuto e adatto ai nostri scopi. Il programma client verrà scritto in C++ per Symbian, sia per fare un po' di pratica su quanto detto, sia perché la maggior parte delle funzionalità del telefono sono accessibili solo tramite il sistema operativo. Allo stato attuale Java supporta le *PIM API (Personal Information Management API)* solo sui modelli più recenti. Sul PC invece utilizzeremo Java, che a fronte di una diminuzione di prestazioni ci offre facilità d'utilizzo. Necessaria quindi la trattazione delle API bluetooth tramite il linguaggio della Sun.

IL CLIENT

Date le sue funzionalità, il client non avrà bisogno di un'interfaccia grafica articolata, un semplice output per il logging dovrebbe bastare. Saranno indispensabili almeno due comandi, uno per la connessione al PC e l'altro per l'invio dei dati. Per semplicità si è scelto di operare su un software già esistente, *btpointtopoint* presente tra gli esempi dell'SDK. Esso implementa una connessione bluetooth tra due dispositivi, risparmiandoci la fatica di scrivere il relativo codice. Il meccanismo è praticamente identico a quello di cui abbiamo parlato nel precedente articolo su bluetooth, ed in più vengono utilizzati gli Active Object, che descriveremo brevemente per avere un'idea del flusso del programma.

COME INIZIARE

1 **Installiamo gli strumenti che ci servono: Nokia SDK 1.2, Microsoft Visual C++ 6.0 e ActivePerl.** Durante l'installazione di Visual C++ ricordiamoci di spuntare la casella per il settaggio delle variabili d'ambiente.

2 **Scompattiamo il contenuto del file *BluetoothContacts.rar* nella stessa partizione nella quale abbiamo installato l'ambiente di sviluppo.** Così facendo evitiamo di andare a toccare alcuni file di configurazione.

3 **Compiliamo il Client.** Apriamo una console dei comandi, entriamo in *BluetoothClient\group* e diamo i due comandi *bldmake bldfiles* e *abld build thumb urel*. Spostiamoci in *BluetoothClient\sis* e digitiamo *makesis btpointtopoint.pkg* per creare l'installativo.

4 **Scarichiamo e istalliamo le librerie necessarie:** Java Communication Api da http://java.sun.com/products/javacomm/index.jsp e JavaBluetooth da www.javabluetooth.org.

5 **Compiliamo il Server.** Se il *CLASSPATH* è settato correttamente, da console posizioniamoci nella directory *BluetoothServer* e digitiamo *javac *.java*. Lanciamo l'applicazione con *java –cp. BluetoothServer.*

6 **Installiamo il Client sul telefonino.** Eseguiamolo. Dal menu in basso a sinistra selezioniamo *"Connetti"*. Carichiamo il PC di casa dalla lista dei dispositivi. Selezioniamo *"Manda Contatti"*. I contatti si trovano adesso sul PC.

Aggiungeremo quindi la fase di lettura dei contatti e l'invio al PC, rimanendo in ascolto di un numero telefonico sul socket bluetooth, per effettuare la chiamata.

IL FLUSSO DEL CLIENT

Aprendo il client contenuto nel CD e dando un'occhiata alla directory dei sorgenti notiamo subito la presenza di svariati file. Quattro di essi vengono generati in automatico dall'*Application Wizard* presente nell'SDK. *Client .cpp* è il nostro riferimento principale. I rimanenti file contengono le funzioni per l'accesso al bluetooth che utilizzeremo. Il punto di partenza è sempre rappresentato dalla funzione *HandleCommandL(TInt aCommand)* della classe che gestisce l'interfaccia utente. Tale funzione è infatti richiamata quando viene selezionata una voce dal menu dei comandi. Il blocco *switch,* interno alla funzione, seleziona l'azione da compiere in base al numero del comando.
Vediamone un frammento:

```
case ECommandConnect:
    iClient->ConnectL();
    break;

case ECommandSendContacts:
    iClient->SendContactsL();
    break;
```

Tutti i comandi sono numerati da una struttura *enum,* all'interno del file *inc/nomeprogramma .hrh. iClient* è un puntatore ad un oggetto della classe *CClient,* che riempiremo con le funzioni interessate. Prima di iniziare a scrivere tale classe, facciamo nostro un concetto molto importante in ambiente Symbian.

GLI ACTIVE OBJECT

Quando in C++ per Symbian incontriamo qualcosa di questo tipo

```
void Write(TDes8& aDes,TRequestStatus& aStatus);
```

significa che siamo di fronte ad una chiamata a funzione asincrona. Essendo asincrona, il flusso del programma continuerà con l'istruzione successiva, mentre la funzione viene eseguita dal relativo thread. Come gestiamo il ritorno da tale funzione? Esistono due modi:

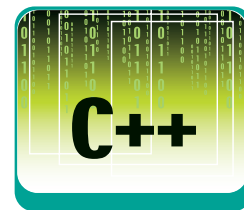
1. Usare **User::WaitForAnyRequest***(aStatus)*
2. Usare gli **Active Object**

Nel primo caso attendiamo volutamente che la chiamata termini, proprio come per una funzione sincrona. Nel secondo invece sfruttiamo una grande facilitazione offerta da Symbian. Per usare gli Active Object dobbiamo estendere la classe astratta *CActive,* e di conseguenza implementare la funzione astratta *RunL()* in essa contenuta. Il meccanismo è il seguente: dopo l'invocazione di una chiamata asincrona, quando tale chiamata termina la sua esecuzione, il controllo passa a *RunL()* che prende le sue decisioni.
Per comunicare a *CActive* che è in corso una richiesta usiamo *SetActive().*

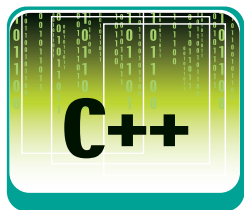
CLIENT.CPP

Tale classe verrà istanziata dall'interfaccia grafica che, guidata dai comandi impartiti dall'utente, richiamerà in sequenza le due funzioni *ConnectL()* e *SendContactsL(),* che qui risiedono.
La fase di costruzione richiede l'inizializzazione di due classi, la prima per la ricerca dei servizi bluetooth e la seconda per l'ascolto sul socket in modalità server.

```
void CClient::ConstructL() {
   iServiceSearcher = CMessageServiceSearcher
                                       ::NewL(iLog);
   User::LeaveIfError(iSocketServer.Connect());
}
```

L'uso del modello *"costruzione in due fasi",* di cui abbiamo parlato in precedenza, impone che *CostructL()* venga richiamata all'interno di *NewLC().* Connettersi ad un device bluetooth si traduce nel richiamare una comoda funzione della classe *CMessageServiceSearcher.*
Dobbiamo inoltre controllare che un'azione corrispondente non sia già stata avviata.

IL CELLULARE È ANCHE TELECOMANDO

Esistono in rete numerose applicazioni che sfruttano la tecnologia bluetooth per i più svariati usi. Tra queste

spicca *Bemused*. Si tratta di un software open source suddiviso in due componenti: un modulo server per PC ed un programma client per Symbian OS. Una volta configurato ed avviato il server, *Bemused* controlla da remoto una playlist di file multimediali, riproducendoli con Winamp o con Windows Media Player. Sono presenti le classiche funzionalità di un player: play, pause, riproduzione a schermo intero e perfino la possibilità di spegnere il PC. Infine l'interfaccia grafica supporta l'utilizzo di skin esterne, di cui se ne può ammirare un esempio in figura. Il sito internet ufficiale è consultabile al seguente url: http://bemused .sourceforge.net/.

Ci viene in aiuto la variabile privata *iState* che assume i valori contenuti nell'apposita *enum TState*, definita in *Client.h:*

```
void CClient::ConnectL() {
   if (iState == EWaitingToGetDevice && !IsActive()) {
        iState = EGettingDevice;
        iServiceSearcher->
                         SelectDeviceByDiscoveryL(iStatus);
        SetActive();
   } else ...
}
```

SetActive() informa l'Active Object che un'operazione è in corso. Quanto il task termina, il flusso prosegue da *RunL()* che aggiorna *iState* in funzione dello stato corrente ed effettua la connessione al device appena selezionato. Questo lavoro è compiuto dalla seguente porzione di blocco *switch:*

```
case EGettingService:
iLog.LogL(_L("Servizio Trovato"));
    iState = EGettingConnection;
    ConnectToServerL();
    break;
```

ConnectToServerL() effettua la connessione sul socket remoto tramite la funzione *Connect():*

```
iSendingSocket.Connect(TBTSockAddr address,
                                TRequestStatus iStatus);
```

La presenza di *iStatus* ci fa capire che si tratta di una chiamata asincrona. Prevediamo quindi che *RunL()* sia in esecuzione quando sono disponibili nuovi dati sul socket d'ingresso. Più precisamente attenderemo un numero di telefono da inoltrare alla funzione *DialNumberL(const TDesC& aPhoneNumber)* che si preoccuperà della composizione. Terminata la fase di connessione dal menu comandi è ora possibile selezionare la voce *"Manda Contatti"*, causando l'invocazione di *SendContactsL()*. Di ogni contatto verranno prelevati dal database il nome e il numero di telefono e tali informazioni verranno scritte sul socket:

```
   TDes8* string;
   TBuf<256> buffer;
   _LIT8(KTagName,"<c>");
TPtrC name = card->CardFields()
                         [nameField].TextStorage()->Text();
buffer.Copy(KTagName);
buffer.Append((const TDesC16&)name);

CnvUtfConverter::ConvertFromUnicodeToUtf8(
                                        *string,buffer);
iSendingSocket.CancelRead();
if (IsActive()) Cancel();
iState = ESendingMessage;
iMessage = (*string).AllocL();
iSendingSocket.Write(*iMessage, iStatus);
SetActive();
```

Per consentire al server di effettuare il parsing dei dati ricevuti, aggiungeremo una stringa *"<c>"* prima di ogni nome ed una stringa *"<n>"* prima di ogni numero di telefono. Infine una stringa *"<e>". iSendingSocket.Write()* richiede, oltre al solito oggetto *TRequestStatus*, un oggetto di classe *HBufC8*, ovvero un buffer allocato nello *Heap*, ovvero in memoria.
Dal momento che le stringhe recuperate dal database dei contatti sono codificate a 16 bit, effettuiamo una conversione da *unicode* a *UTF8* tramite l'apposita classe *CnvUtfConverter* e poi allochiamo la memoria con *AllocL()*. Terminato il ciclo di scrittura, dovremmo aver trasferito tutti i contatti.

PREPARIAMO IL PC PER IL SERVER

Supposto che possediate un dispositivo bluetooth e che esso sia installato correttamente sul vostro PC, procuriamoci ciò che ci serve. Su sistemi operativi Microsoft Windows il gestore Bluetooth permette di interfacciare qualunque software con i driver hardware di basso livello tramite le comuni porte seriali. Per consentire alla nostra applicazione java il trasferimento dei dati dobbiamo procurarci sia la libreria per l'accesso alla porta seriale, sia l'implementazione dello stack Bluetooth.
Scarichiamo dunque le *Java Communication Api* dall'indirizzo *http://java .sun.com/products /javacomm/index.jsp* e installiamole. Dobbiamo effettuare tre operazioni: includere *comm.jar* nella variabile d'ambiente *CLASSPATH*, copiare *win32com.dll* nella directory *JAVA_SDK/bin*, copiare *javax.comm.properties* in *JAVA_SDK /jre /lib*. *JAVA_SDK* è il percorso di installazione dell'SDK Java. Un'implementazione open source dello stack bluetooth può essere scaricata dal sito *www.javabluetooth.org*. L'installazione richiede di aggiungere al *CLASSPATH* l'intera directory delle classi.

PROGETTIAMO IL SERVER

Modularizziamo il lavoro e suddividiamo il codice in tre classi: la GUI, il server bluetooth e la

classe per la gestione di ogni contatto. Amministrare una connessione bluetooth utilizzando Java non presenta particolari difficoltà.
Cominciamo innanzitutto dalla classe *BluetoothServer*, la quale contiene il metodo *main*. Essa dovrà estendere la classe *UARTTransport*, contenuta nella libreria *javabluetooth*, ed effettuare l'overriding dei metodi *void receiveData(byte[] packet, int length)* e *void receiveData(byte[] packet, int length)*, che ci consentiranno di leggere e scrivere sul socket stringhe codificate in byte.
Il main effettuerà due operazioni, inizializzare lo stack bluetooth e creare l'interfaccia grafica.
Vediamo un po' di codice:

```
public static final void main(String[] args) throws
                                        Exception {
  BuetoothServer server = new BluetoothServer(
                                        "COM3");
  HCIDriver.init(server);
  hciDriver = HCIDriver.getHCIDriver();
  GUI gui = new GUI();
}

public BluetoothServer(String serialPort) throws
                                        Exception {
  super(serialPort);
  singleton = this;
}
```

La costruzione della superclasse *UARTTransport* necessita di una porta seriale come parametro d'ingresso, proprio perché, tramite le *Java Comm API*, essa comunica con il gestore bluetooth installato sul PC.

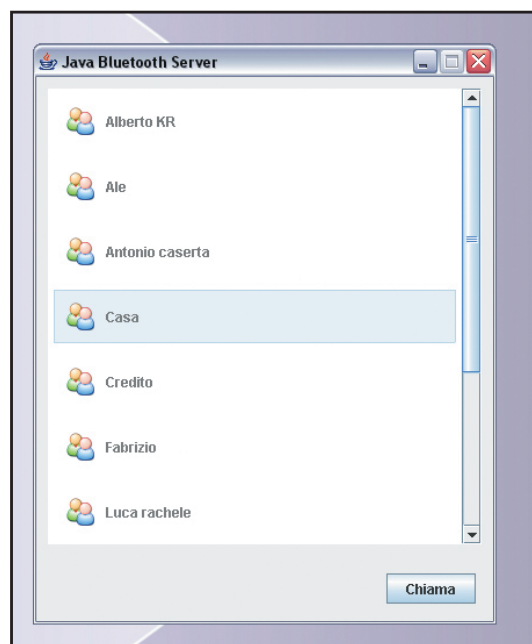



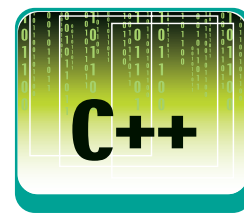
Fig. 1: L'interfaccia in java sarà estremamente linerare

L'oggetto appena creato è poi passato al metodo init della classe *HCIDriver* che ci configura come interfaccia primaria verso il socket bluetooth. Inoltre viene istanziata la *GUI*.
UARTTransport ci comunica i nuovi dati in ingresso non appena essi sono disponibili.
La funzione *receiveData* dovrà effettuare il corretto parsing sui tag *"<c>"*, *"<n>"* ed *"<e>"*, costruire un oggetto *Contact* con le corrispondenti informazioni ed aggiungere il contatto appena creato alla GUI.
La **Figura 1** ritrae il risultato di tale computazione. La pressione del bottone *"Chiama"* scatena l'invocazione del metodo *actionPerformed* sul proprio ascoltatore, ricavando il contatto selezionato dall'utente e inviandolo al dispositivo bluetooth che funge da client.

```
Contact contact = (Contact)list.getSelectedValue();
  if (contact != null) {
    StringBuffer sb = new StringBuffer("            ");
int length = contact.getNumber().length();
if (length <= 16) sb.replace(
                    0,length,contact.getNumber());
try {
  BluetoothServer.singleton.sendBTPacket(
                sb.toString().getBytes("UTF-16LE"));
} catch (Exception ex) { ex.printStackTrace(); }
```

CONCLUSIONI

Il corso su Symbian OS si chiude qua. Spero che quest'ultimo articolo abbia suscitato in tutti voi la curiosità di programmare il proprio telefonino. Una cosa è certa, di approfondimenti se ne possono fare tanti. Una piccola base per iniziare rimane sempre un buon punto di partenza.
Buon coding a tutti.

Antonio Trapani

FACCIAMO ATTENZIONE ALLE LIBRERIE

**Durante la stesura di un programma per Symbian OS capita spesso di ottenere strani errori in fase di compilazione.
Un progetto è sempre suddiviso in numerosi file ed una pratica comune consiste nel lavorare esclusivamente sui sorgenti, tralasciando ogni altra cosa.
Il file .mmp, contenuto nella directory group di ogni progetto, possiede informazioni rilevanti riguardo i file sorgenti e le librerie usate.
Prendendo un caso specifico, se utilizziamo le API per i contatti e per il bluetooth dobbiamo includere le seguenti righe:**

```
LIBRARY    esock.lib
LIBRARY    bluetooth.lib
LIBRARY    btextnotifiers.lib
LIBRARY    btmanclient.lib

LIBRARY    sdpagent.lib
LIBRARY    sdpdatabase.lib
LIBRARY    cntmodel.lib
```

La documentazione Symbian, a corredo con l'SDK, tiene traccia, per ogni classe, della relativa libreria. Ricordiamoci quindi di aggiungerla al progetto!

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

JAVA

LEGGERE UN FILE DI CONFIGURAZIONE DA JAVA

Spesso sentiamo l'esigenza di rendere configurabili una serie di parametri nelle nostre applicazioni J2SE, come indirizzi internet, port numbers, valori predefiniti, nomi di file, etc. Un modo semplice e veloce per leggere questi valori ci è offerto dalla classe di sistema *java.util.Properties*. Con i suoi metodi la classe indicizza il contenuto di un *file properties* (che è un semplicissimo file di testo) in una mappa di coppie chiave = valore. Il tip illustra una classe che estende *java.util.Properties* da utilizzare subito nelle nostre applicazioni. Una volta istanziata la classe basterà utilizzare chiamate del tipo *Configuration.getProperty ("chiave", "valore_di_default");* per leggere i parametri di configurazione. Si rimanda alla documentazione della classe base per maggiori informazioni sui metodi disponibili.

```
package it.domtest.utils;
import java.io.IOException;
/**
 * Una semplicissima estensione della classe java.util.Properties per
 * l'utilizzo come file di configurazione.
 *
 * @author  Domenico Testa <domenico.testa@gmail.com>
 */
public class Configuration extends java.util.Properties {
    /** Nome del file (fully qualified) di configurazione di default */
    private static final String CONFIG_FILE_NAME = "
                                                 /conf/application.properties";
    /**
     * Crea una nuova istanza della classe.
     * Legge il profilo di configurazione nel file di properties di default.
     */
    public Configuration() throws IOException {
        this(CONFIG_FILE_NAME);
    }
    /**
     * Crea una nuova istanza della classe.
     * Legge il profilo di configurazione nel file di properties specificato.
     *
     * @param file nome del file di configurazione da leggere
     */
   public Configuration(String file) throws IOException {
      load(getClass().getResourceAsStream(CONFIG_FILE_NAME));
   }
    /**
     * Salva il file di configurazione in uno stream di output.
     *
     * @param out stream di output destinazione del file di configurazione
     */
    public void save(java.io.OutputStream out) throws IOException
    {
        store(out, "File di configurazione generato automaticamente
                                   dalla classe it.domtes.utils.Configuration");
    }
}
```

JAVA SCRIPT

REALIZZARE FORM STANDARD

Questo tip vuole segnalare un'interessantissima libreria open source per la realizzazione di form web dinamiche dall'alto contenuto interattivo, utilizzando solamente meccanismi e tecnologie standard, oggi supportate dalla maggior parte dei browser.

La libreria in questione si chiama *wForms* e la si può liberamente scaricare dal sito *http://www.formassembly.com/* dove è presente, tra le altre cose, anche una web application che permette di generare dei *wForms* in maniera visuale.

Una volta costruita la nostra form le sole operazioni da seguire sono: pubblicare i file di supporto sul nostro sito, copiare il codice XHTML generato dal formbuilder nella nostra pagina e aggiungere i seguenti riferimenti nella sezione *HEAD* del documento:

```
<link rel="stylesheet" href="wforms.css"  type="text/css" />
<link rel="alternate stylesheet" href="wforms-jsonly.css"  type=
                    "text/css" title="stylesheet activated by javascript" />
<script type="text/javascript" src="wforms.js" ></script>
<script type="text/javascript" src="js/localization-it.js" ></script>
```

In allegato alla rivista trovate un esempio di form costruito utilizzando tale strumento.

C#

LEGGERE IL MAC ADDRESS DELLA SCHEDA DI RETE

Questo semplice tip illustra come interrogare il sistema operativo per conoscere gli indirizzi *MAC (Media Access Controll address)* delle schede di rete installate sul computer. Questa informazione risulta utilissima per quelle reti wireless in cui bisogna comunicare tale informazione all'amministratore della rete per abilitare al connettività. Chiaramente l'indirizzo *MAC* è solo un pretesto per illustrare il semplice l'utilizzo del namespace *System.Management* messo a disposizione dal .NET Framework per ottenere informazioni sull'ambiente hardware e software in cui girano le nostre applicazioni .NET.

```
using System;
using System.Text;
using System.Runtime.InteropServices;
using System.Management;
namespace ListMacs
{  /// <summary>
   /// Questa semplice applicazione mostra come elencare gli indirizzi MAC
   /// di tutte le schede di rete installate sulla macchina.
   /// </summary>
   class MACLister
   {  /// <summary>
      /// Il punto di ingresso principale dell'applicazione.
      /// </summary>
      [STAThread]
      static void Main(string[] args)
      { PrintMACAddresses(); }
      public static void PrintMACAddresses()
      { ManagementClass mc = new ManagementClass(
                                   "Win32_NetworkAdapterConfiguration");
        ManagementObjectCollection moc = mc.GetInstances();
        string MACAddress  = String.Empty;
        string description = String.Empty;
        foreach(ManagementObject mobj in moc)
        { if((bool)mobj["IPEnabled"] == true)
          {  MACAddress = mobj["MacAddress"].ToString();
             description = mobj["Description"].ToString();
             Console.WriteLine("Scheda " + description + ": " +
                                                      MACAddress); }
          mobj.Dispose(); }
      }
   }
}
```

IL TIP DEL MESE

ACCEDERE ALLA RUBRICA DI OFFICE

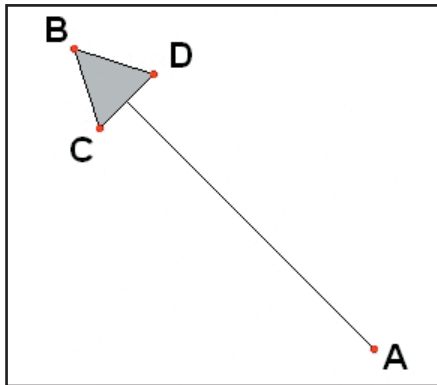
In molte applicazioni, soprattutto quelle di tipo CRM, si sente la necessità di gestire una rubrica dei contatti per inviare email, recuperare indirizzi e numeri di telefono.
Spesso, quindi, ci ritroviamo a riscrivere l'ennesimo modulo di gestione contatti quando, nella maggior parte dei casi, il nostro cliente ha già installato Microsoft Outlook per gestire la sua rubrica. Perché allora rinunciare alle funzionalità di una rubrica professionale nella nostra applicazione? In questo tip utilizziamo COM per accedere alla rubrica di Outlook. Chiaramente, perché il codice funzioni correttamente, sulla macchina deve essere installato il pacchetto Microsoft Office.

```
Imports System.Reflection
Module ContactList
  Sub Main()
    ' Istanzia l'applicazione (Outlook):
    Dim outlookApp As Outlook.Application = New
                                    Outlook.Application()
    Dim oNS As Outlook.NameSpace =
                         outlookApp.GetNamespace("mapi")
    ' Passare i dati corretti per il login in questa funzione:
    oNS.Logon("Outlook", Missing.Value, False, True)
    ' Recupera la lista dei contatti:
    Dim cContacts As Outlook.MAPIFolder = oNS
     .GetDefaultFolder(Outlook.OlDefaultFolders.olFolderContacts)
    Dim oItems As Outlook.Items = cContacts.Items
    Dim oContact As Outlook.ContactItem
    Try
      Dim i
      For i = 1 To oItems.Count
        oContact = oItems.Item(i)
        ' Stampa alcune proprietà:
        Console.WriteLine(oContact.FullName)
        Console.WriteLine(oContact.Email1Address)
        ' Oppure visualizza il contatto nella sua finestra:
        oContact.Display(True)
      Next
    Catch
      Console.WriteLine("si è verificato un errore durante le
                                   lettura del contatto")
    Finally
      ' Logout:
      oNS.Logoff()
      ' Clean up delle risorse:
      outlookApp = Nothing
      oNS = Nothing
      oItems = Nothing
      oContact = Nothing
    End Try
  End Sub
End Module
```

Disegnare le frecce

State lavorando ad un progetto che contiene molte linee di collegamento tra oggetti e vorreste dare un verso a tali linee?
Se la risposta è sì, allora avete bisogno di una bella freccia! Il codice relativo a questa operazione è davvero semplice da comprendere e veloce da applicare.
Può essere riadattato a qualsiasi linguaggio di programmazione o framework.Il nostro esempio si basa sulle funzioni grafiche di wxWidgets ed è scritto in c++.Vedremo, inoltre, come riadattare il codice per riutilizzarlo in OpenGL.Per realizzare il seguente esempio è stato utilizzato l'IDE freeware Dev-c++ con il package wxWidgets installato.

Stefano Vena

‹1› COS'È UNA FRECCIA

Possiamo vedere una freccia come un insieme di quattro punti connessi tra loro.
L'ordine di connessione (seguendo la figura) è A,B C D e ancora B.
In particolare verrà tracciata una linea da A a B e poi verrà disegnato il poligono BCD.

‹2› LE PRIME RIGHE DI CODICE

```
wxPaintDC dc(this);
      ...
float alfa , cosalfa , sinalfa;
wxPoint p[3];
float altezza=10;
float larghezza = 5;
wxPoint A(300 , 300 ) ;
wxPoint B ( 100 , 100 );
```

Se ad esempio ci troviamo a gestire l'evento *Paint* di wxWidget per prima cosa otteniamo il riferimento al device contex poi andiamo a definire qualche variabile di supporto. Tra le variabili dichiarate c'è un vettore utilizzato per disegnare la testa della freccia, la larghezza e l'altezza della stessa testa.Infine, impostiamo il punto di partenza e di arrivo della nostra freccia.

‹3› PREPARIAMO LA TESTA

```
alfa=atan2(( A.y - B.y ), ( A.x - B.x ) );
cosalfa = cos( alfa );
sinalfa = sin( alfa );

//Punto B della testa
p[0]= B;

//Punto C
p[1].x = p[0].x + (int)(
                altezza * cosalfa
               -larghezza * sinalfa
                );
p[1].y = p[0].y + (int)(
                altezza  * sinalfa
               +larghezza * cosalfa
                );
//Punto D
p[2].x = p[0].x + (int)(
                altezza  * cosalfa
               +larghezza * sinalfa
                );
p[2].y = p[0].y + (int)( altezza * sinalfa
               -larghezza *cosalfa
                );
```

Non c'è che dire, il codice parla da solo! Calcoliamo la pendenza della freccia(*alfa*). Di questa angolo calcoliamo seno e coseno e andiamo ad assegnare alle celle del vettore i valori corretti.

‹4› DISEGNAMO LA FRECCIA

Ora viene la parte migliore.
Ecco le ultime righe di codice necessarie per disegnare una freccia.

```
dc.SetPen( *wxBLACK);
dc.SetBrush( *wxBLACK);

dc.DrawLine(A,B);
dc.DrawPolygon(3, p, 0, 0, wxWINDING_RULE);
```

Per prima cosa andiamo a scegliere il colore da utilizzare per la coda e lo associamo alla penna corrente attraverso *SetPen(x)* poi associamo il colore della testa al *Brush* corrente attraverso *SetBrush.* Una volta compiuta questa operazione andiamo a disegnare prima la coda e poi la testa della freccia.
Ovviamente facciamo uso delle primitive che abbiamo fin qui sviluppato.
Il codice non presenta alcuna difficoltà concettuale. Tutta la complessità é legata al calcolo delle coordinate dei punti A e B, oltre che del punto P. Calcolo che abbimo eseguito nei passaggi precedenti

‹5› LO STESSO CODICE IN OPENGL

```
float Ax = 0.0f; float Ay = 0.0f;
float Az = 0.0f; float Bx = 5.0f;
float By = 5.0f; float Bz = 0.0f;
float altezza= 0.1f;
float larghezza = 0.05f;
float alfa = atan2( Ay - By, Ax - Bx );
float cosalfa = cos( alfa ); float sinalfa = sin( alfa );
glColor3f( 1.0f,0.0f,0.0f); glBegin( GL_LINES );
 //Punto A
 glVertex3f( Ax,Ay,Az)
 //Punto B
 glVertex3f( Bx,By,Bz)
glEnd();
glBegin( GL_TRIANGLE_STRIP );
//Punto B
glVertex3f( Bx,By,Bz)
//Punto C
glVertex3f(Bx + altezza * cosalfa - larghezza * sinalfa,
     By + altezza * sinalfa + larghezza * cosalfa,0.0f);
 //Punto D
glVertex3f(Bx +  altezza * cosalfa +  larghezza *sinalfa,
      By +  altezza * sinalfa -  larghezza * cosalfa, 0.0f);
//Punto B per la chiusura
glVertex3f( Bx,By,Bz)    GL.glEnd();
```

‹6› COMMENTI

Per quanto il codice sia semplice, la soluzione proposta non è sicuramente ottima. Infatti, sia la prima versione per wxWidgets, sia la seconda per OpenGL mostrano alcuni punti deboli dal punto di vista computazionale.
Il richiamare molte volte le funzioni seno, coseno e atan2 può rallentare molto il processo se le frecce da disegnare sono molto numerose. Inoltre, molte variabili possono essere dichiarate una sola volta ed al di fuori della routine di disegno.
Se si fa uso di qualche accortezza ,che comunque va oltre gli obiettivi di questo articolo, si possono ottenere buoni risultati.

Realizzare un generatore di numeri casuali

A volte capita di dover lavorare a programmi che richiedono l'utilizzo dei numeri casuali.
Il linguaggio c++ ci offre il supporto alla generazione di tali numeri, però noi non ne abbiamo il controllo.
In questo semplice express andremo a vedere come generarli.
Non ci occuperemo solo di generarli, ma andremo anche a gestirli tramite alcune distribuzioni statistiche. In particolare utilizzeremo le distribuzioni uniforme, esponenziale, normale o gaussiana e di poisson.
In questa sede non verranno trattate le caratteristiche teoriche di ognuna delle distribuzioni.
Per realizzare il seguente esempio possiamo utilizzare l'ambiente di sviluppo (freeware) Dev-C++, scaricabile gratuitamente dal sito web : *www.bloodshed.net*

Stefano Vena

‹1› LE PRIME RIGHE DI CODICE

```
#ifndef CRANDOMGENERATOR_H
#define CRANDOMGENERATOR_H
class CRandomGenerator
{   private:
            static const int m;
            static const int a;
            static const int q;
            static const int r;
          int seed;
```

Per prima cosa creiamo due file il primo di nome *"rnd.h"* ed un secondo di nome *"rnd.cpp"* nel file con estensione *"h"* andiamo a scrivere il codice del primo passo. Gioca un ruolo importante la variabile *seed* (seme). Infatti essa viene utilizzata per dare il via alla generazione della sequenza di numeri casuali. Per evitare che due istanze diverse della classe *CRandomGenerator* diano come risultato lo stesso valore, condividiamo i parametri *m, a, q, r* tra tutte le classi dichiarandole *static.* A questo punto andiamo a vedere la dichiarazione dei membri della classe e poi procediamo con l'implementazione.

‹2› I METODI

```
public:
  CRandomGenerator(); CRandomGenerator(int seed);
 ~CRandomGenerator();
  void NewSeed(int seed = -1);
  double random();
   double uniform(double lo, double up );
  double exponential( double lamda );
  double normal(double mean,double std);
  double poisson( double lamda ); };
#endif // CRANDOMGENERATOR_H
```

La funzione *NewSeed* cambia il seme del generatore. *random* invece ritorna un numero casuale compreso tra 0 e 1 di tipo *double.* La funzione *uniform* ritorna un numero casuale compreso nell'intervallo *"lo"* - *"up"*.

‹3› I PRIMI PASSI

```
#include "rnd.h"
#include <math.h>
const int CRandomGenerator::m =
                    2147483647;
const int CRandomGenerator::a=16807;
const int CRandomGenerator::q=m/a;
const int CRandomGenerator::r=m%a;
CrandomGenerator::
          CRandomGenerator(int seed)
{
  this->seed = seed;
}
CRandomGenerator::CRandomGenerator()
{
  seed = 314159;
}
CRandomGenerator::~CRandomGenerator(){}
```

Ora passiamo all'implementazione dei metodi.
Per prima cosa andiamo a definire il valore delle variabili statiche e i costruttori. La variabile *m* viene inizializzata con il valore limite del range dei numeri interi a 32 bit. Alle altre variabili vengono assegnati dei valori ricavati da operazioni al limite del range di validità. Il compilatore non dà errore ma i registri del sistema verrano riempiti con dei valori casuali.

‹4› LA GENERAZIONE

```
void CRandomGenerator::NewSeed(int seed)
{ if (seed  < 0)
            this->seed = (int) time (NULL);
  else
            this->seed = seed; }
double CRandomGenerator::random()
{ int tmpseed = a*(seed%q)-r*(seed/q);
   if ( tmpseed > 0 ) seed = tmpseed;
    else seed = tmpseed + m;
   return (double)seed*(1.0/m); }
```

Queste due funzioni sono le più importanti. La funzione *NewSeed* serve a cambiare il seme del generatore. Se il valore passato è negativo, la funzione assocerà un valore in base al tempo attuale. Le operazioni del metodo random sono incomprensibili ma funzionano!

‹5› LE DISTRIBUZIONI

```
double CrandomGenerator::
       uniform( double lo, double up )
{ return (up-lo)*random()+lo;}
double CrandomGenerator:: exponential( double lamda )
 { return -(1/lamda)*log( random() ); }
double CrandomGenerator::normal(double mean, double std)
 {  double v1, v2, s, rnn1, y;
    for(;;){
      v1=2.0*random()-1.0;  v2=2.0*random()-1.0;
      s=v1*v1+v2*v2;
      if( s < 1.0 ) break; }
    rnn1=v1*sqrt( -2.0*log(s)/s );  y=mean+rnn1*std;
    return y; }
 double CrandomGenerator::poisson( double lamda )
 { double x=0.0, a=exp( -lamda ), s=1.0;
   for(;;){
      s=s*random();
      if( s<a ) break;
      x += 1.0; }
  return x;
 }//poissob
```

‹6› UTILIZZIAMO LA CLASSE

```
CRandomGenerator rnd;
      rnd.NewSeed( 56524 );
 cout << rnd.random()  << endl;
 cout << rnd.uniform( 0 , 10 )  << endl;
 cout << rnd.exponential( 2 )  << endl;
 cout << rnd.normal( 5, 0.014 ) << endl;
 cout << rnd.poisson( 0.5 ) << endl;
```

Questa è una carrellata delle funzionalità della classe appena creata.

mIRC: Un primo approccio

Tra i vari linguaggi di programmazione, possiamo citare Visual Basic, Delphi, Java, e tanti altri, ma se riflettiamo per un istante alla difficoltà riscontrata dalla maggior parte degli utenti telematici nell'avere un approccio con la programmazione, torna molto utile riferirsi a mIRC.mIRC, come molti sanno è un client utilizzato per "chattare", ma quello che non tutti sanno è che ha uno "script editor" integrato, che permette tramite un proprio linguaggio, di operare sulle funzionalità del client. Ovviamente ci si riferisce ad eventi, variabili locali e globali, condizioni, cicli, alias e quanto altro occorre per iniziare così a conoscere le basi della "Creazione". Avendo uno script editor che lavora esclusivamente in funzione del programma stesso, non si possono creare eseguibili o librerie, ma "addons" testuali che vengono caricate poi nel client mIRC.

Danilo Lucirino

‹1› FUNZIONALITÀ DEGLI EVENTI. L'ON <AZIONE>

Gli eventi si riferiscono a particolari azioni effettuate dagli utenti che utilizzano mIRC. Nel momento in cui un utente scrive un messaggio, entra in funzione l'*on text,* quando uscirà da un "canale" entrerà in funzione l'*on part,* quando effettua una disconnessione dal server ecco l'*on quit,* e così via. Ad ogni azione verrà effettuato il comando specificato nell'editor.

‹2› IDENTIFICATORI

Vengono utilizzati degli identificatori, come *$chan* per il canale, *$nick* per il nickname usato dall'utente o *$server* per il server a cui si è connessi. Tramite l'utilizzo degli alias possono essere creati altri identificatori che potrebbero tornare "necessari" per il raggiungimento del nostro progetto. In questo caso abbiamo creato l'identificatore *$oradata*. Digitando */prova* avremo in echo: *ORA* del *DATA* (Es. 16:00 del 11/05/2005

‹3› VARIABILI

Le variabili possono essere *locali* o *globali,* come *"tag"* utilizzano il simbolo *%*.
Per intendere una variabile *globale,* che rimarrà costante fino al prossimo cambiamento, all'interno del client, useremo */set %nomevariabile.* Per specificare invece una variabile *locale,* che lavorerà solo nell'esecuzione di un determinato codice per poi svanire nel nulla, useremo */var %nomevariabile.*

‹4› LE CONDIZIONI

Per utilizzare le condizioni si fa solitamente riferimento a *IF, ELSEIF* ed *ELSE.* Le condizioni vengono specificate tra (e). Nel codice d'esempio, digitando */controlla,* avremo il risultato *ON* solo se la variabile impostata corrisponde ad *ON.* Nel caso in cui così non fosse, ci ritornerà il testo *ERRORE.* Con un po' di dimestichezza avremo le giuste basi per addentrarci nel mondo della programmazione.

‹5› ON JOIN, ESEMPIO DI SEGNALAZIONE DEL SERVER

Alcuni eventi ci vengono segnalati dal server su cui siamo connessi. È difatti il server a dirci quando un utente entra in un canale, quando cambiano le modalità di un canale, o quando comunque qualcosa "accade" senza un avviso diretto da parte dell'utente. Con la stringa nell'esempio, saluteremo pubblicamente l'utente, appena sarà entrato nel canale, tramite il comando */msg $chan Ciao $nick ! Benvenuto in $chan!* Avremo così usato l'evento on join, il comando */msg* e gli identificatori *$chan* per il canale e *$nick* per l'utente.

‹6› PROGRAMMARE CON "NOTEPAD"

Ciò che scriviamo all'interno dello script editor di mIRC, non è altro che semplice e puro testo, che non dobbiamo preoccuparci di compilare, in quanto mIRC utilizza gli scripts, o *addons,* basandosi su semplici documenti di testo. Potremmo infatti creare le stringhe del nostro programma anche con Notepad, per poi caricare il file da mIRC, con il comando */load –rs "nome del file"* (l'opzione *–rs* indica che stiamo caricando uno script nei remotes di mIRC). Il formato pertanto è indifferente, anche se le estensioni più usate per questi scripts sono *.ini* e *.mrc.*

Una piccola classe Variant

In alcuni casi ci troviamo a dover lavorare con degli oggetti che possono essere di natura diversa, ma che vorremmo gestire con una sola struttura dati. Se gli oggetti in questione sono tutti derivati dalla stessa classe padre, allora non abbiamo problemi ad utilizzare il polimorfismo. Ma se dovessimo "mescolare" tipi di base (int,float...) la situazione inizierebbe a peggiorare. Il problema si risolve utilizzando un tipo di dati *variant*, ovvero un tipo particolare capace di incapsulare al suo interno qualsiasi tipo di dato.
In questo express vedremo come realizzare una piccola classe variant in grado di gestire interi, numeri in virgola mobile e stringhe. La classe è concepita in modo tale da poter essere utilizzata anche come parser per lo scambio di dati attraverso un socket.
Per realizzare l'esempio abbiamo bisogno di un compilatore c++, un editor di testo e un po di dimistichezza con il c++.
Come al solito consiglio l'utilizzo dell'IDE freeware Dev-c++.

Stefano Vena

‹1› CONCETTI DI BASE

```
#include <cstdlib>

typedef enum
{
 INT    = 0,
 FLOAT  = 1,
 STRING = 2,
 EMPTY  = 3
} VariantType;
```

Per prima cosa dichiariamo una enumerazione da utilizzare per far "capire" alla classe *variant* che tipo di dati sta gestendo.In questo esempio daremo la possibilità di gestire interi di tipo *long*, valori in virgola mobile di tipo *double*, e stringhe come puntatori di caratteri. Definiamo anche la voce *EMPTY* per gestire il caso di oggetti vuoti. Inoltre includiamo la *cstdlib.*

‹2› I PRIMI PASSI

```
#define reserve(VAR_TYPE,BASE_TYPE,COUNT)\
      size = sizeof(BASE_TYPE)*COUNT;\
      data = malloc( size );\
      type = (data != NULL) ?
                VAR_TYPE: EMPTY
#define Cast(To) (*((To*)data ))
```

La prima cosa che facciamo è definire alcune utili macro che aumenteranno la leggibilita del codice in seguito. La macro reserve serve ad allocare lo spazio necessario per la memorizzazione dei nostri dati. L'utilizzo di tale macro è semplice; essa riceve un valore *VariantType* il nome del tipo di dati di base corrispondente ed il numero di variabili da memorizzare. Il numero di variabili è sempre uno tranne che nel caso delle stringhe dove corrisponde al numero di caratteri più uno. La macro *Cast* serve per fare il cast della variabile di supporto alla classe che stiamo realizzando e la tratteremo più in là.

‹3› SI ENTRA NEL VIVO

```
class variant
{
  private:
   void * data;
   size_t size;
   VariantType type;
```

La variabile *data* è il cuore della classe essa viene utilizzata per il salvataggio dei dati.
La variabile *size* invece conserva la dimensione in byte della variabile gestita da "data".
L'ultima variabile invece conserva il tipo di dati gestito. Anche in questo caso il codice é autoesplicativo e non presenta alcuna difficoltà. Molto semplicemente abbiamo gettato le basi per potere utilzzare in un secondo momento il tipo *variant.*
La maggiort parte del llavoro é fatto non ci resta che inizializzare la classe e poi creare le strutture per gestirla al meglio.

‹4› I COSTRUTTORI

```
public:
variant( long value )
{  reserve( INT , long , 1 );
    if( type == INT ) Cast(long) = value; }
variant( double value )
{  reserve( FLOAT , double , 1 );
    if( type == FLOAT ) Cast(double) = value; }
variant(const char* value )
{ reserve(STRING,char, strlen( value) + 1 );
   if( type == STRING )
 {
   strcpy( (char*)data , value );
  }
}

~variant(){ if( data != NULL) free(data); }
```

L'interpretazione dei costruttori è davvero banale. Ognuno di essi riserva la memoria necessaria a gestire un particolare tipo di dato e se l'operazione va a buon fine fa il cast della memoria appena allocata per poter memorizzare il valore iniziale.

‹5› LA GESTIONE

```
VariantType getTipo(){ return type; }
    size_t getSize(){return size; }
double getFloat(){ return  type == FLOAT ?
                   Cast(double) : -1; }
long   getInt(){ return  type == INT ?
                 Cast(long) : -1; }
const char* getString(){
                 return  type == STRING ?
                 (char*)data  : ""; }
bool setFloat( double value )
{   if( type != FLOAT ) return false;
  Cast(double) = value;
  return true; }
bool setInt( long value )
{  if( type != INT ) return false;
 Cast(long) = value;
 return true; }
operator double () {return getFloat(); }
operator long () {return getInt(); }
operator const char*(){return getString();}
```

‹6› L'UTILIZZO

```
int main(int argc, char *argv[])
{
    variant Float(10.5);
    variant Int(10L);
    variant String("Io Programmo");

    Int.setInt( Int.getInt() + 10 );

    cout << (double)Float << endl;
    cout << Int.getInt() << endl;
    cout << (const char*)String << endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Ecco un breve esempio per poter osservare la classe all'opera. La flessibilità del metodo appare evidente.

Realizzare una classe C++ per controllare la connessione in Rete

Frequentemente le nostre applicazioni richiedono una connessione ad internet per funzionare correttamente. Come fare per sapere se il nostro computer è connesso e, in caso contrario, come possiamo stabilire una connessione? Windows ci mette a disposizione la libreria wininet.dll che esporta una serie di funzioni di alto livello che fanno al caso nostro. L'unica difficoltà è quella di imparare le funzioni giuste ed rispettare le convenzioni di chiamata necessarie ad utilizzarle. Proviamo a capitalizzare la nostra esperienza in una classe C++ che esponga un'interfaccia intuitiva e che possiamo riutilizzare nelle nostre applicazioni senza dover spulciare continuamente MSDN.

Per realizzare questo esempio è stato utilizzato l'ambiente di sviluppo Microsoft Visual Studio .NET ma quanto esposto può essere implementato utilizzando un qualsiasi compilatore per windows.

Domenico Testa

<1> L'INTERFACCIA DELLA CLASSE

```
#ifndef INTERNETCONNECTION_H
#define INTERNETCONNECTION_H
namespace domtes {
  class CInternetConnection {
    public:
      CInternetConnection();
      bool connect();
      bool disconnect();
      bool hasRAS() const;
      bool isConfigured() const;
      bool isConnected() const;
      bool isOffline() const;
      bool usesLAN() const;
      bool usesModem() const;
      bool usesProxy() const;
    private:
      unsigned long dwStatusFlags;
  };
}
#endif // INTERNETCONNECTION_H
```

A noi serve una classe che semplicemente ci dica se sul sistema esiste una connessione ad internet configurata correttamente, se è attualmente connesso e, se necessario, stabilisca la connessione. L'interfaccia della nostra classe rispecchia in pieno questi requisiti. In aggiunta ci facciamo suggerire dalle wininet una serie di predicati booleani che ci daranno maggiori informazioni sulla nostra informazione ad internet, cioè se è una connessione tramite modem, tramite LAN, se viene usato un proxy e se sul sistema è installato RAS.

<2> IL COSTRUTTORE

Il costruttore inizializza la variabile membro dwStatusFlags con i flag di stato restituiti dalla funzione InternetGetConnectionStatus().
Questa variabile deve essere considerata come una maschera di bit contenente tutte le informazioni sulla nostra connessione.

<3> IL METODO CONNECT()

```
bool CInternetConnection::connect() {
  bool retval = InternetAutodial(INTERNET_
                    AUTODIAL_FORCE_ONLINE, NULL);
  InternetGetConnectedState(&dwStatusFlags, 0);
  return retval;}
```

Il metodo non fa altro che richiamare la funzione InternetAutodial() per attivare la connessione ad internet predefinita del sistema. Essendo una funzione che altera lo stato della connessione, la variabile dwStatusFlags viene reimpostata interrogando la funzione InternetGetConnectedStatus()

<4> IL METODO DISCONNECT()

```
bool CInternetConnection::disconnect() {
  bool retval = InternetAutodialHangup(0);
  InternetGetConnectedState(&dwStatusFlags, 0);
  return retval; }
```

Il metodo disconnect() è il duale del metodo connect(). Chiude la connessione aperta ed aggiorna le informazioni sullo stato della connessione.

<5> IMPLEMENTAZIONE DEI PREDICATI

```
bool CInternetConnection::isConfigured() const
{ return dwStatusFlags &
                INTERNET_CONNECTION_CONFIGURED;
}
bool CInternetConnection::isOffline() const
{ return dwStatusFlags &
                  INTERNET_CONNECTION_OFFLINE;
}
```

In ogni predicato andiamo semplicemente a testare il valore del bit corrispondente alla proprietà che vogliamo controllare nella maschera dwStatusFlags.

<6> UTILIZZIAMO LA CLASSE

```
#include <iostream>
using namespace std;

#include "InternetConnection.h"

int main(int argc, char* argv[])
{
  domtes::CInternetConnection internet;

  cout << "Stato connessione: " << endl;
  cout << std::boolalpha;
  cout << "Configurata:  \t" << internet.isConfigured() << '\n'
  << "Connesso:  \t" << internet.isConnected()  << '\n'
  << "Usa un modem:  \t" << internet.usesModem()  << '\n'
  << "Usa la LAN:    \t" << internet.usesLAN()    << '\n'
  << "Usa un proxy:  \t" << internet.usesProxy() << '\n'
  << "RAS installato:\t" << internet.hasRAS()    << '\n';
  // Se necessario prova a connettersi ad internet
  if(internet.isConfigured() && !internet.isConnected())
  {
    cout << "Tentativo di connessione...";
    if(internet.connect())
    {
     cout << "OK\n";
     // Dunque si disconnette:
     cout << "Disconnessione...";
     if(internet.disconnect())
     {
       cout << "OK\n";
     } else { cout << "FALLITO\n";
     }
   } else
   {
     cout << "FALLITO\n";
   }
  }
  return 0;
}
```

Il codice implementa un semplice programma che, utilizzando la nostra classe, interroga lo stato della nostra connessione ad internet e prova i metodi per la connessione e la disconnessione.

mIRC: un primo approccio

Tra i vari linguaggi di programmazione, possiamo citare Visual Basic, Delphi, Java, e tanti altri, ma se riflettiamo per un istante alla difficoltà riscontrata dalla maggior parte degli utenti telematici nell'avere un approccio con la programmazione, torna molto utile riferirsi a mIRC.mIRC, come molti sanno è un client utilizzato per "chattare", ma quello che non tutti sanno è che ha uno "script editor" integrato, che permette tramite un proprio linguaggio, di operare sulle funzionalità del client. Ovviamente ci si riferisce ad eventi, variabili locali e globali, condizioni, cicli, alias e quanto altro occorre per iniziare così a conoscere le basi della "Creazione". Avendo uno script editor che lavora esclusivamente in funzione del programma stesso, non si possono creare eseguibili o librerie, ma "addons" testuali che vengono caricate poi nel client mIRC.

Danilo Lucirino

<1> FUNZIONALITÀ DEGLI EVENTI. L'ON <AZIONE>

Gli eventi si riferiscono a particolari azioni effettuate dagli utenti che utilizzano mIRC. Nel momento in cui un utente scrive un messaggio, entra in funzione l'*on text,* quando uscirà da un "canale" entrerà in funzione l'*on part,* quando effettua una disconnessione dal server ecco l'*on quit,* e così via. Ad ogni azione verrà effettuato il comando specificato nell'editor.

<4> LE CONDIZIONI

Per utilizzare le condizioni si fa solitamente riferimento a *IF, ELSEIF* ed *ELSE.* Le condizioni vengono specificate tra (e). Nel codice d'esempio, digitando */controlla,* avremo il risultato *ON* solo se la variabile impostata corrisponde ad *ON.* Nel caso in cui così non fosse, ci ritornerà il testo *ERRORE.* Con un po' di dimestichezza avremo le giuste basi per addentrarci nel mondo della programmazione.

<2> IDENTIFICATORI

Vengono utilizzati degli identificatori, come *$chan* per il canale, *$nick* per il nickname usato dall'utente o *$server* per il server a cui si è connessi. Tramite l'utilizzo degli alias possono essere creati altri identificatori che potrebbero tornare "necessari" per il raggiungimento del nostro progetto. In questo caso abbiamo creato l'identificatore *$oradata.* Digitando */prova* avremo in echo: *ORA* del *DATA* (Es. 16:00 del 11/05/2005

<5> ON JOIN, ESEMPIO DI SEGNALAZIONE DEL SERVER

Alcuni eventi ci vengono segnalati dal server su cui siamo connessi. È difatti il server a dirci quando un utente entra in un canale, quando cambiano le modalità di un canale, o quando comunque qualcosa "accade" senza un avviso diretto da parte dell'utente. Con la stringa nell'esempio, saluteremo pubblicamente l'utente, appena sarà entrato nel canale, tramite il comando */msg $chan Ciao $nick ! Benvenuto in $chan!* Avremo così usato l'evento on join, il comando */msg* e gli identificatori *$chan* per il canale e *$nick* per l'utente.

<3> VARIABILI

Le variabili possono essere *locali* o *globali,* come *"tag"* utilizzano il simbolo *%.*
Per intendere una variabile *globale,* che rimarrà costante fino al prossimo cambiamento, all'interno del client, useremo */set %nomevariabile.* Per specificare invece una variabile *locale,* che lavorerà solo nell'esecuzione di un determinato codice per poi svanire nel nulla, useremo */var %nomevariabile.*

<6> PROGRAMMARE CON "NOTEPAD"

Ciò che scriviamo all'interno dello script editor di mIRC, non è altro che semplice e puro testo, che non dobbiamo preoccuparci di compilare, in quanto mIRC utilizza gli scripts, o *addons,* basandosi su semplici documenti di testo. Potremmo infatti creare le stringhe del nostro programma anche con Notepad, per poi caricare il file da mIRC, con il comando */load –rs "nome del file"* (l'opzione *–rs* indica che stiamo caricando uno script nei remotes di mIRC). Il formato pertanto è indifferente, anche se le estensioni più usate per questi scripts sono *.ini* e *.mrc.*

SOFTWARE SUL CD

ITEXT 1.3.1

Il pdf è fatto

iText è una libreria OpenSource scritta in Java che consente di dotare le nostre applicazioni di funzionalità di esportazione verso il formato PDF. Si tratta di una libreria decisamente importante tanto che gli dedichiamo un articolo in questo stesso numero di ioProgrammo. Il PDF è un formato universale che garantisce una grande precisione nella stampa e una portabilità senza precedenti. Il poter creare documenti in formato PDF da una nostra applicazione sicuramente ne innalza il valore. D'altra parte itext non solo gestisce la mera esportazione dei dati in formato PDF ma dispone di funzionalità specifiche per crittografare o firmare digitalmente i documenti così creati, si tratta perciò di una libreria particolarmente interessante che risolve una volta per tutte il problema della generazione di report o di stampe sofisticate

Directory /iText

PBEANS 1.3.1

Persistenza dei dati senza problemi

Ne parliamo in questo stesso numero nel bell'articolo di Daniele De Michelis. Pbeans è una libreria scritta in Java che consente di implementare la persistenza dei dati nelle nostre applicazioni. Grazie a Pbeans normali tabelle, colonne e campi di un database relazionale possono essere trattate come oggetti e classi, inoltre lo stato dell'applicazione può essere salvato in un database. Si tratta sicuramente di un bel vantaggio. Non ha certo velleità di sostituire Hibernate o Castor, tuttavia è un buon tool, rapido ed efficiente da usare in progetti di dimensioni tali da non giustificare l'uso di strumenti così complessi.

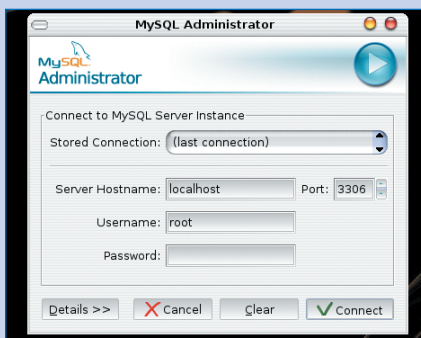
Directory /Pbeans

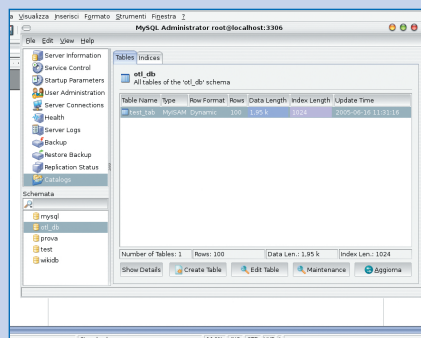
MYSQL ADMINISRATOR

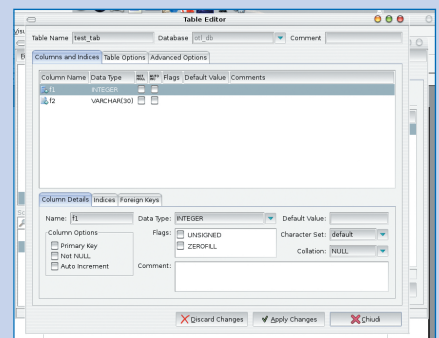
Il cervello di MySQL

Un buon database non si basa solo sulle performance o sulle funzionalità , ma deve anche essere semplice da amministrare, la dove per amministrazione si intente la creazione di nuovi utenti, la gestione dei permessi sulle tabelle, la creazione delle repliche e il tuning del sistema. MySQL Administrator è tutto questo. Una potente interfaccia grafica in grado di gestire ogni particolare del vostro sistema di DB. Si tratta di uno strumento importante, essenziale che sta portando MySQL ad essere conosciuto anche da un certo numero di programmatori abituati a fare i conti con le interfacce grafiche prima che con il backend del sistema. MySQL per sua natura è completamente gestibile da linea di comando, ma certamente una bella GUI rende tutto più semplice ed immediato.

Directory: /MySQL Administrator

› CONNESSIONE

1 Avvia MySQL Administrator ed inserisci i parametri di connessione al tuo server di database

› TABELLE

2 Seleziona la tabsheet catalogs, scegli il database, la tabella da modificare e clicca su edit table

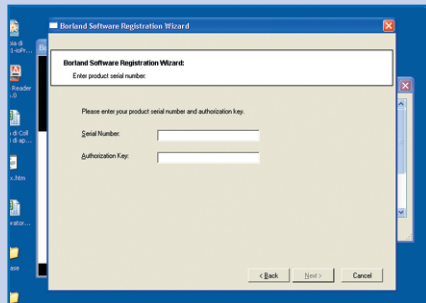
› MODIFICHE

3 Clicca con il tasto destro ed esegui modifiche o inserimenti sulle colonne che ti servono

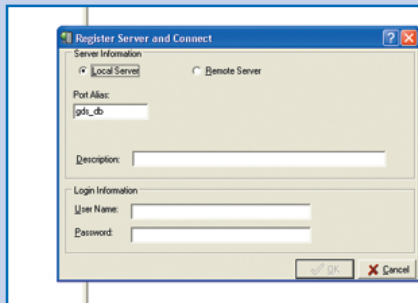
BORLAND INTERBASE 7.5

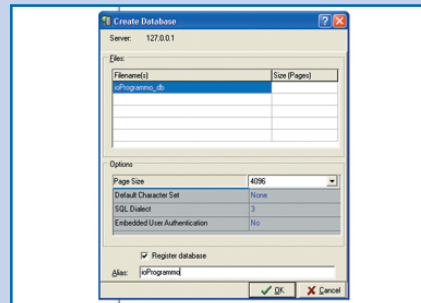
Il db che ha fatto la storia

Nato con le prime versioni di Delphi, compatibile SQL, multiDb, multithreading quando ancora tutti usavano db personalizzati e i più evoluti al massimo Access, Interbase ha fatto la storia della programmazione. Rilasciato sotto licenza OpenSource nel periodo in cui Borland è diventata Imprise salvo poi ri-tornare un software commerciale in tempi successivi è senza dubbio un database dalle prestazioni straordinarie, che probabilmente non è diventato un leader solo a causa dell'altalenanza di situazioni in cui la sua politica di Marketing si è venuta sviluppare. Resta comunque un prodotto straordinario decisamente da provare.

Directory /Interbase

› INSTALLAZIONE

1 L'installazione di interbase 7.5 non comporta particolari difficoltà. È necessario registrarsi presso *http://www.borland.com/downloads /download_interbase.html* per ottenere una licenza trial. Vi verrà inviata una email contenente un file di testo da copiare nella home di installazione del prodotto.
All'atto dell'installazione quando vi verrà chiesto di registrarvi potete semplicemente cliccare su Cancel.

› AMMINISTRARE

2 Dopo avere installato il prodotto, nel menu di windows, nel percorso start/borland interbase 7.5 server/ troverete l'utility ibconsole. Lanciatela e cliccate su "interbase server /register" nella parte alta sinistra dell'interfaccia.
Vi verrà proposta una maschera in cui inserire le password per l'amministrazione del database.
I vostri dati sono SYSDBA/masterkey come utente local.

› IL PRIMO DATABASE

3 Cliccate con il pulsante destro del mouse sull'icona Database e di seguito su create database. Nella finestra di dialogo che vi viene proposta inserite il nome del database nella prima riga e non dimenticate di inserire un alias.
Quando avrete terminato cliccate su ok lasciando un check sul flag "register database". Il database verrà creato e nella colonna sinistra vedrete i vari oggetti che lo compongon.

MYSQL QUERY BROWSER

Il braccio di MySQL

Se da un lato MySQL Administrator consente di gestire tutte le impostazioni del server, dall'altro non offre un'interfaccia efficace verso la programmazione SQL. Questo tipo di supporto è demandato a MySQL Query Browser che è un tool studiato in maniera specifica per immettere query SQL, gestire tabelle, righe e colonne. Si tratta del software che eredita tutte le funzionalità del vecchio MySQL Control Center non più sviluppato e che è stato smembrato nei due progetti *MySQL Administrator* e *Query Browser*, e dobbiamo dire che si tratta di una scelta azzeccata vista l'alta specializzazione raggiunta da questi due strumenti nello svolgere i loro compiti

Directory /mysql query browser

MYSQL CONNECTOR

Il ponte verso i linguaggi

Un database da solo non sarebbe niente se non fosse possibile sviluppare applicazioni che consentano di immagazzinare e manipolare i dati.
Le applicazioni come sanno certamente i nostri lettori sono fatte dai linguaggi di programmazione.
Per cui deve esistere qualcosa per collegare i linguaggi al database, per offrire ai linguaggi funzionalità di accesso specifiche e ottimizzate per i database, nel caso di MySQL questo software "ponte" prende il nome di *connectors*, in questo numero presentiamo quello per Java e quello per .NET, per PHP il *connectors* è già integrato nel linguaggio, per tutti gli altri c'è ODBC il driver universale per i database.

Directory /MySQL Connector

OSCOMMERCE 2.2

Il commercio elettronico a portata di click

In molti affermano che il commercio elettronico è solo un mito mai realizzato. Noi non siamo di questo avviso, lo dicevamo già qualche anno fa, se affiancato a una linea commerciale tradizionale il commercio elettronico è uno straordinario strumento che apre nicchie di mercato che altrimenti sarebbero difficilmente raggiungibili. Così non c'è azienda che non disponga almeno di una piccola vetrina dei suoi prodotti online.
OSCommerce è molto più che un sistema per creare una vetrina, è un completo quanto complesso sistema di creazione di siti di commercio elettronico. Scritto in PHP con l'ausilio di MySQL accompagna l'utente lungo tutta la pipeline dell'ordine

portandolo dal click sul prodotto fino a pagare alla cassa gli oggetti acquistati. I gateway di pagamento supportati sono tantissimi, così come la pipeline dell'ordine e la disposizione degli oggetti, la creazione dei cataloghi è altamente personalizzabile. L'unico difetto che riconosciamo a OSCommerce è una certa difficoltà nella modifica dei template, non eccessiva ma non degna dell'attenzione all'usabilità con cui è stato sviluppato il resto del sistema.

Directory /OSCommerce

DRUPAL 4.6.1

Il costruttore di blog

E così internet è giunta all'epoca dei blog, i piccoli siti personali su cui pubblicare in modo semlice dei contenuti grazie a un minicms. Drupal è un sistema scritto in PHP tale che ciascun utente iscritto al sistema diventi automaticamente l'amministratore di un blog.

Drupal è particolarmente orientato in questo senso, la versione che vi presentiamo contiene inoltre alcune interessanti funzionalità in materia di personalizzazione e atomizzazione delle risorse. A ciascun blog è adesso possibile associare un file di configurazione separato dagli altri, stabilire a quali database farà riferimento e molto altro ancora. Il proprietario di ciascun blog adesso può anche personalizzare i temi indipendentemente da quelli generali. Un passo avanti per Drupal che sta diventano il riferimento per sistemi di questo tipo e rivaleggia ormai ad armi pari con il più blasonato *Moveable Type* e con i più anziani *WordPress* e *NucleusCMS*.

Directory: /Drupal

PHPMYADMIN 2.6.3

Il controllore di MySQL

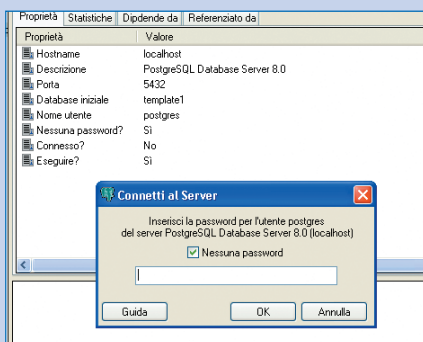
PHPMyAdmin potrebbe occupare senza troppe preoccupazioni anche la categoria Database. Non perché esso stesso un database ma perché si tratta di una Web Application che offre un completo frontend verso MySQL. Come Web Application viene installata su sistemi remoti e la sua maggiore utilità si avverte in sistemi di Hosting quan-

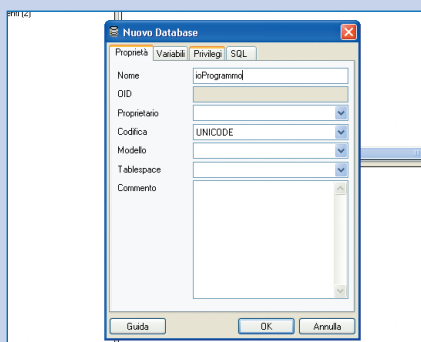
POSTGRESQL 8.0.3

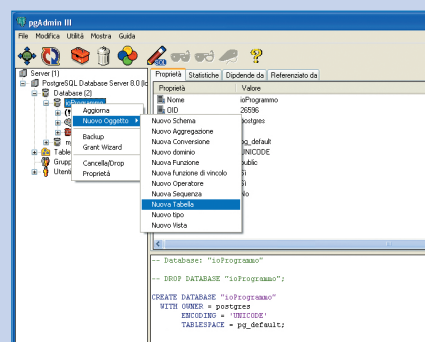
Il più completo

Per certi versi PostgreSQL ha subito nel corso del tempo gli attacchi sferrati dai concorrenti MySQL e Firebird per primi, tuttavia ha conservato intatta tutta la sua base di installato. I motivi di questa stabilità sono insiti nelle caratteristiche del database: solido come una roccia, veloce oltre ogni aspettativa Postgres è usatissimo sia in applicazioni web che in applicazioni standalone. Probabilmente fino a qualche tempo fa soffriva del fatto di essere multipiattaforma ma con un occhio speciale rivolto a Linux, il che lo rendeva particolarmente complicato da installare in ambienti Windows. Sembrerebbe che questa limitazione sia stata ampiamente superata, quindi anche i programmatori Windows possono godersi la potenza di questo incredibile server di database.

Directory /PosteGreSQL

› IL LOGIN

1 L'interfaccia di amministrazione di PostgreSQL è pgAdmin. Subito dopo l'installazione la troverete come di consueto disponibile nel meni di Windows. Lanciatela, selezionate il server a cui volete connettervi. Nella maschera di login inserite l'utente Postgres con la password che avete scelto all'atto dell'installazione. Una volta loggati tutti gli oggetti del server compariranno nel menu a sinistra

› IL PRIMO DATABASE

2 Cliccate con il tasto destro del mouse su "Postgres SQL Server 8.0" e dal menu che comparirà scegliete "nuovo oggetto" e di seguito "nuovo database". L'unica informazione essenziale da inserire per la creazione è il nome del database. Se avete già creato degli utenti o avete bisogno di settare particolari permessi per un gruppo o per un utente specifico potete farlo dalla tabsheet "privilegi"

› LA PRIMA TABELLA

3 Come di consueto sarà sufficiente cliccare sullo schema "pubblic" e selezionare con il tasto destro "nuova tabella". Tutte le successive operazioni possono essere eseguite dalla finestra di dialogo che vi comparirà. Potrete settare eventuali ereditarietà della tabella, aggiungere le colonne con i vari tipi, i vincoli e i privilegi. PostgreSQL offre moltissime opzioni ed un'elevata flessibilità.

do si vuole fornire ai propri utenti una completa interfaccia di gestione del database MySQL senza per questo costringerli a installare applicazioni di terze parti. Nonostante questo PHP-MyAdmin viene ormai installato anche in ambienti di sviluppo in sostituzione delle tradizionali interfacce standalone a causa delle sue enormi potenzialità della facilità di utilizzo e della quantità di funzioni che offre.

Directory /phpmyadmin

FIREBIRD 1.5.2.4

L'araba Fenice dei database

Firebird è un server di database nato dal codice sorgente di Interbase rilasciato da Imprise Corporation prima di tornare ad essere Borland.

È una storia un po' complicata, quello che è importante sapere è che si tratta di un database molto affidabile nato da codice che tuttora sta alla base di Borland Interbase e che presentiamo in questo stesso numero, con in più l'importante vantaggio di essere completamente free e perciò gratuito. Dal punto di vista delle caratteristiche tecniche, il database gira correttamente sia su Linux che su Windows, offre una compatibilità quasi completa con l'ANSI SQL-99ed ha prestazioni che lo rendono particolarmente appetibile per quanti hanno bisogno di un database leggero, potente, affidabile e gratuito.

Directory: /FireBird

FIREBIRD .NET PROVIDER

Per usare Firebird in .NET

Chi lo dice che i progetti OpenSource prolificano in ambiente Linux e tralasciano l'ambiente Windows? Firebird non ha affatto dimenticato i programmatori Microsoft ed ha sviluppato questo .NET Provider che mette in grado chi usa Visual Basic, C# e qualunque altro linguaggio sotto il cappello del framework .NET di sviluppare applicazioni che sfruttino in maniera nativa e perciò con prestazioni ottimali il database Firebird.

Directory: /Firebird Net Provider

HSQL DB 1.8.0

Il database dal peso piuma

Ne facciamo uso nel bell'articolo di Daniele de Michelis su Pbeans presente in questo stesso numero di ioProgrammo. Si tratta di un database interamente scritto in Java la cui caratteristica principale è quella di essere estremamente leggero, oltre che multipiattaforma. Se avrete la pazienza di leggere l'articolo su Pbeans scoprirete come HSQL DB sia stato sapientemente utilizzato come storage di persistenza, offrendo ad un'applicazione java un supporto incredibilmente omogeneo alle caratteristiche del linguaggio

Directory: HSQLDB

MYSQL 4.1.1.2

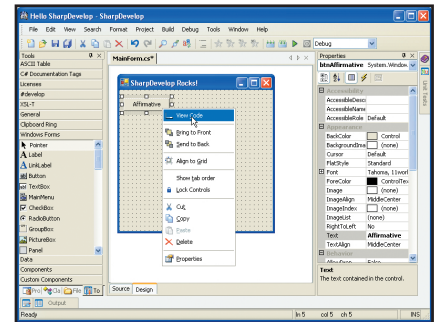
Lo scheletro di internet

Così tanti sono i progetti su internet che fanno capo a MySQL che si può tranquillamente dire che MySQL è una delle strutture portanti del Web. Nato come DB ultraleggero per servire progetti di piccole dimensioni si è evoluto nel tempo fino a diventare uno dei database con il maggior numero di funzionalità esposte. Si va dalla ricerca full text al supporto alle transazioni senza per questo dimenticare la leggerezza del sistema. MySQL è estremamente semplice da usare e al contempo estremamente potente e questo ne fa uno dei server di database più usati al mondo. Qualcuno ancora lo taccia di non essere sufficientemente professionale da reggere il confronto con i DB commerciali più costosi, noi non siamo fra questi. MySQL è del tutto paragonabile a server di DB del costo di svariate migliaia di euro e in molti casi prevale in prestazioni e affidabilità.

Directory: MySQL

SHARPDEVELOP 1.1.10

L'alternativa a Visual Studio

Se siete programmatori C# ma non volete utilizzare i costosi ambienti di Microsoft, SharpDevelop si rivela un'ottima alternativa. Si stratta di un ambiente leggero, visuale, rad dotato di tutte o quasi tutte le facilities di ambienti molto più costosi, ma dotato dell'interessante caratteristica di essere Free e privo di costi.

Se credete che questo possa essere sinonimo di scarsa affidabilità sappiate che state commettendo un grave errore, SharpDevelop è decisamente un ambiente professionale, ben studiato e progettato, inoltre il team di sviluppo è particolarmente attivo.

Directory /Csharp

APACHE 1.3.33/2.0.54

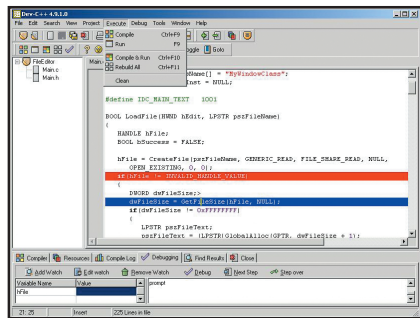
Il Web Server tuttofare

La maggioranza dei programmatori Windows molto probabilmente sarà abituata a fare i conti con IIS in fase di progettazione delle applicazioni. È anche vero che in fase di produzione soprattutto per quanto riguarda applicazioni web che saranno mantenute in vita grazie a sistemi di Hosting, molto probabilmente troverete una larga disponibilità di sistemi Apache +Linux e una disponibilità più ridotta di sistemi Windows. Apache è un ottimo Web Server. Estremamente leggero, potente in grado di servire un quantitativo incredibile di linguaggi e sistemi. Si rivela un grande strumento sia in fase di progettazione del codice che in fase di produzione, potete installarlo tranquillamente in ambiente Windows e si rivelerà un grande supporto per tutte quelle applicazioni che non fanno uso di ASP o di .NET. In questo caso potrete ancora usare Apache ma a costo di qualche configurazione non troppo pulita. In ogni caso rimane un software da installare, un indispensabile per tutti coloro che sviluppano applicazioni Web.

Directory: /Apache

DEV C++ 4.9.9.2

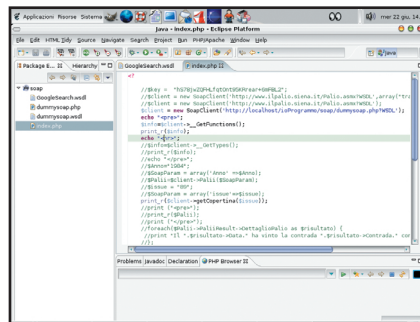
Uno standard in ambiente Windows

I programmatori C++ sono abituati a gestire ogni cosa direttamente senza mezzi termini e senza intermediari.
Chi programma C++ sa benissimo che la forza del linguaggio sta nell'estrema flessibilità e nella grande libertà di decidere quale livello di astrazione il programmatore vuole avere nei confronti del sistema.
Dev C++ in breve tempo è diventato l'ambiente di riferimento per chi programma in C++, rivaleggia quanto a diffusione con il più quotato e senza dubbio più ricco di funzionalità Microsoft Visual C++.

L'assenza di eccessivi fronzoli in questo caso non è uno svantaggio, il programmatore C++ si trova ad usare un prodotto maturo, dotato di tutte le funzionalità essenziali, capace di produrre eseguibili con diversi compilatori. Inoltre l'ambiente è estensibile grazie ai DevPack. Molto probabilmente vorrete usarlo, se avete intenzione di approcciare in qualunque modo la programmazione C++

Directory: /DevCPP

ECLIPSE 3.1

L'editor tutto fare

Cento MB, questa è la dimensione raggiunta ormai dal mitico ambiente tuttofare che sta facendo il bello e il brutto tempo fra i programmatori Java. Non che la dimensione sia sempre sinonimo di qualità anzi spesso non lo è, ma nel caso di Eclipse si può tranquillamente fare un'eccezione. Questo IDE orientato alla programmazione Java, ma completamente estensibile per mezzo di plugin per essere utilizzato con quasi ogni linguaggio esistente esporta un numero talmente elevato di caratteristiche che l'occupazione dello spazio sull'Hard Disk è pienamente giustificata. Meno giustificata è una certa pesantezza dell'ambiente, se non per il fatto che il tool stesso è scritto in Java che notoriamente per alcuni tipi di applicazioni non è certo un mostro di velocità.

In ogni caso se avete un sistema sufficientemente dotato in termini di risorse sicuramente Eclipse è un ambiente che vi toglie le castagne dal fuoco in più di una situazione. Per numero di funzionalità si può paragonare a quello che è stato Emacs in tempi non troppo lontani.

Directory /Eclipse

JAVA DEVELOPMENT KIT 1.5.0 UPGRADE 3

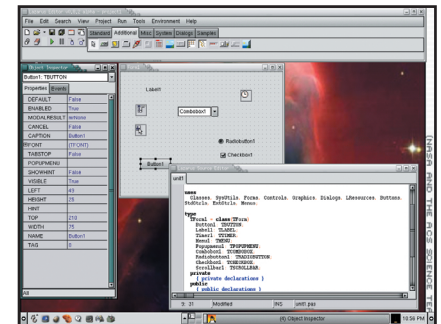
Indispensabile per programmare in Java.

Ci corre l'obbligo, prima di tutto, di fare i nostri migliori auguri a java che ha appena compiuto 10 anni di vita. Il linguaggio di Sun nato con l'ambizione di essere il primo realmente multipiattaforma e che portava con se la grande ambizione di servire qualunque tipo di periferica, dopo 10 anni di esistenza può dire di avere largamente realizzato i suoi obiettivi. Con una base di programmatori larghissima e con il primato invidiabile di essere il linguaggio base per i telefonini di nuova generazione come per i PC come per i sistemi embedded è ad oggi uno dei linguaggi più diffusi al mondo. Per programmare in Java è necessario semplicemente dotarsi del J2SE che vi presentiamo in questo stesso numero, tutto il resto sono AddON, anche se non neghiamo che un buon editor aiuta. Non vi resta che installare il JDK dotarvi di entusiasmo e pazienza e creare il vostro primo "Hello Word" carta di ingresso del meraviglioso mondo di Java.

Directory /JDK150

LAZARUS 0.9.6

Il clone OpenSource di Delphi

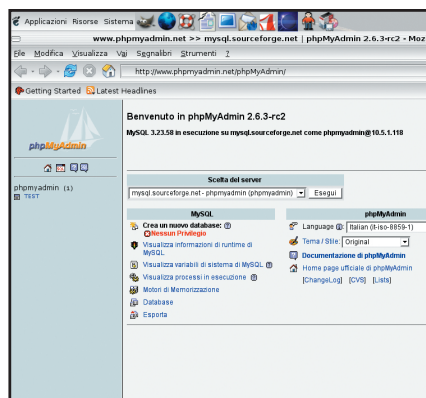
I programmatori più anziani avranno almeno una volta avuto il piacere di programmare in Pascal. Si tratta di un linguaggio meraviglioso, molto didattico e potente.

Il principale ambiente di programmazione che fa capo a Pascal è sicuramente Delphi, che come tutti i progetti di Borland è stato ed è tutt'ora il miglio IDE RAD disponibile sul mercato. Purtroppo allo stato attuale dotarsi di un Delphi 2005 costa un bel po' di denaro e anche se la spesa è assolutamente giustificata dalla bontà del prodotto, in realtà medio piccole qualcuno vorrebbe poter programmare in Pascal usando un ambiente come Delphi ma senza dover prosciugare l'intero fondo cassa. Per queste persone c'è Lazarus un clone OpenSource di Delphi, che supporta ObjectPascal e nelle funzionalità ricorda da vicino uno dei primi Delphi. Per imparare Object pascal, per comprendere la logica della programmazione ad oggetti per capire come funziona la programmazione ad eventi, Lazarus è senza dubbio l'ambiente che fa per voi. Infine ultimo ma non in ordine di importanza, Lazarus è multipiattaforma, ne esistono versioni anche per Linux.

Directory /lazarus

PHP

L'essenza dell'OpenSource

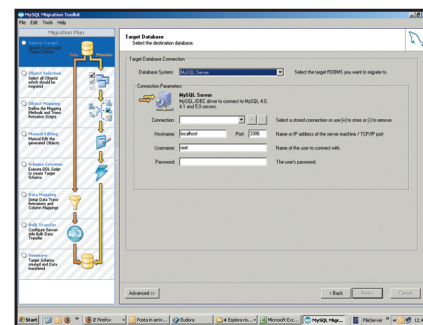
A nostro avviso non c'è linguaggio che abbia dato all'OpenSource la stessa spinta che ha offerto PHP, almeno per quanto riguarda il web.

Nato come linguaggio procedurale si è evoluto nel tempo aggiungendo un completo supporto alla programmazione Object Oriented.
Oggi nella sua versione 5 è probabilmente il linguaggio che più di ogni altro ingloba in maniera nativa un quantitativo straordinario di primitive che da sole consentono di dare concretezza praticamente a qualunque progetto.

Se a tutto questo si aggiunge che PHP è estendibile tramite moduli, che è OpensSource che è multipiattaforma e che è altamente integrato con MySQL ma anche con tutti gli altri database si capisce il perché della fortuna di questo linguaggio. Se avete a che fare con il Web prima o poi avrete a che fare anche con PHP.

Directory/ PHP

MYSQL MIGRATION KIT

Migra i dati a MySQL in un clic

MySQL si sta sempre affermando di più come uno standard de facto nel campo dei database.
Sempre più frequente si manifesta la volontà di passare le proprie applicazioni dall'uso di un particolare DB a MySQL.

Ad esempio è il caso di Access, SQL Server, Oracle, ciascuno dei quali in particolari condizioni si rivela inadeguato per un certo tipo di applicazione.
Access non è multiutente, Oracle è troppo grande per molte applicazioni di taglio Web.
Da oggi la migrazione è semplificata con questo MySQL Migration Kit che con una procedura semplificata consente di passare facilmente da un formato di database ad un altro.

Directory /

Intel C++ compiler

Il compilatore Intel cucito sulle caratteristiche del processore!

Molte aree della programmazione, come il calcolo scientifico, la manipolazione real-time dei segnali e delle immagini e l'Intelligenza Artificiale, hanno in comune la necessità di prestazioni molto elevate. In questi casi una corretta ottimizzazione del codice non è sufficiente a garantire la velocità richiesta. Diventa indispensabile compilare per uno specifico modello di processore, traendo vantaggio dalla possibilità di appoggiarsi su specifiche sicure e avanzate. È in questo momento che entra in gioco il compilatore di Intel ottimizzato per i processori dell'omonima piattaforma, che produce un codice compilato che sfrutta la potenza dei processori fino all'ultimo bit. Attenzione per utilizzare il prodotto è necessario registrarsi sul sito https://registrationcenter.intel.com/EvalCenter/EvalForm.aspx?ProductID=411&lang=en per ottenere la licenza evaluation che vi da diritto a usare il prodotto per 30 giorni. Allo stato attuale è disponibile anche la versione 9 del compilatore, il sito di riferimento è http://www.intel.com/cd/software/products/asmo-na/eng/compilers/219964.htm.

Directory /

Maguma Open Studio 1.0

Maguma Open Studio è una delle poche soluzioni Open Source che può rivaleggiare in quanto a completezza e potenza con Zend Studio.

Si tratta di un IDE, ovvero un ambiente di sviluppo che aiuta il programmatore nella creazione del proprio codice. Le caratteristiche di Maguma Open Studio che semplificano la vita del programmatore sono molteplici. Chiaramente è dotato di Code Completion e Sintax Highlighting che sono sempre la base per ogni IDE che si rispetti. Oltre a queste due caratteristiche comuni a molti ambienti di programmazione ce ne sono molte altre che fanno di Magma un ottimo IDE. Ad esempio la possibilità di gestire direttamente le connessioni PHP e quindi modificare direttamente i file in remoto. Questa possibilità si rivela molto comoda soprattutto quando non si vuole installare un intero ambiente di test sulla propria macchina e si vuole comunque gestire da un solo prodotto tutto il ciclo di sviluppo di un software.

FUNZIONI AGGIUNTIVE

Dal punto di vista più strettamente operative, è interessante annotare la presenza di un Class e Functions Browser tramite il quale è possibile spostarsi fra i punti saliente del codice senza doverlo scorrere interamente. Oltre a questo è anche possibile spostarsi fra i file di include. Queste caratteristiche sono indispensabili soprattutto quando si lavora in progetti di grandi dimensioni che prevedono la suddivisione in molti file di include e molte classi. Utile anche la snippet Library che consente di inserire in una sorta di repository del codice ripetitivo da inserire automaticamente nei propri progetti con un semplice click del mouse.

DEBUGGER

Una delle funzioni più richieste e difficili da trovare in ambienti meno evoluti è quella del debugger. Consente di tracciare lo stato delle variabili a tempo di esecuzione così come inserire dei breakpoint o eseguire il codice in step successivi.

A completare il tutto c'è il comodo preview interno o esterno che consente di visualizzare il sito a cui si sta lavorando in anteprima

CONCLUSIONI

Si tratta di un ottimo prodotto, probabilmente il miglior IDE OpenSource disponibile. Maguma sta compiendo uno sforzo senza precedenti per esportare non solo un prodotto OpenSource ma un'intera filosofia OpenSource che fa della condivisione delle risorse e delle informazioni un tassello importante.

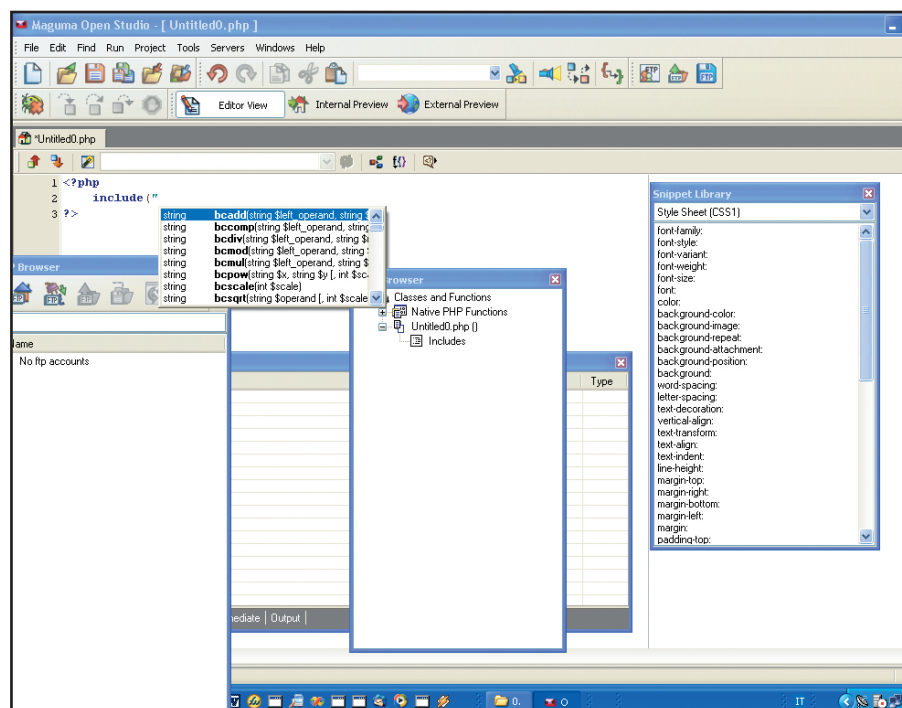



Fig. 1: Una vista d'insieme dell'ambiente

MultiThreading e Regioni Critiche

Ecco come i sistemi operativi affrontano il problema dell'accesso contemporaneo di due o più processi ad un'unica risorsa. Vi sveliamo i segreti della concorrenza parlando di Regioni Critiche e Semafori

Supponete di avere due processi in esecuzione. Il processo *A* tenta di stampare, il processo *B* contemporaneamente esegue lo stesso tentativo. Il risultato? Se non gestito, sarebbe del tutto casuale, blocco della stampante, esecuzione prima del processo *A* poi del processo *B* nel migliore dei casi. Da questo banale esempio si capisce come il problema di gestire risorse condivise sia sentito e importante nella gestione di un sistema operativo. Una risorsa potrebbe essere anche la CPU o la memoria, il non avere garanzie sull'uso condiviso di queste risorse potrebbe essere assolutamente deleterio nella programmazione di sistemi operativi multitasking. Tentiamo prima di ogni cosa di dare qualche definizione:

- **"interferenza"** o **"race condition"** una situazione in cui due processi stiano tentando di accedere contemporaneamente alla stessa risorsa danneggiandosi a vicenda.
- **"regione critica"** una porzione di processo che potenzialmente potrebbe accedere a risorse condivise.

Detto questo dobbiamo trovare un modo per gestire le condizioni di interferenza. Per farlo ci daremo delle regole:

1. Due processi non possono trovarsi contemporaneamente nella loro regione critica.
2. Nessuna previsione deve essere fatta riguardo alla velocità con cui un processo sarà eseguito.

Va da sé che per rispettare queste due regole, processi separati devono poter comunicare fra loro. Ovvero, un processo che entra nella propria regione critica deve comunicarlo agli altri processi di modo che questi si adeguino e non entrino anche essi nella propria regione critica. Per raggiungere il nostro scopo di gestione delle interferenze, aggiungiamo altre due regole

1. Nessun processo fuori dalla sua regione critica può bloccare un altro processo.
2. Nessun processo deve attendere indefinitamente di entrare nella propria regione critica.

I SEMAFORI

L'idea di base è la seguente: un processo prima di entrare nella sua sezione critica esegue una chiamata di sleep e rimane bloccato fino a che un altro processo non lo risveglia con una chiamata di wakeup. La chiamata di wakeup deve essere effettuata al termine dell'esecuzione del codice posto in regione critica. Questo non risolve ancora completamente il problema, una soluzione più interessante è quella proposta da Dijkstra. L'idea è quella di condividere una variabile chiamata semaforo *(semaphore)*. Inizialmente il valore del semaforo sarà 0. L'algoritmo si basa sul seguente tipo:

```
type semaphore=record
  valore:integer;
  l: <lista di processi>;
end;
```

Il tipo contiene un valore e una lista di processi. Nella fase di attesa un processo viene aggiunto alla lista e viene decrementato il valore. La procedura *V* incrementa il valore e rimuove (libera) il processo.

```
procedura P(S:semaphore);
begin
        S.valore:=S.valore-1;
        if S.valore<0
  then begin
     <aggiungi il processo a S.L>
```

```
    sleep
        end;
end;

procedura V(S:semaphore);
begin
  S.valore:=S.valore+1;
          if S.valore<=0
  then begin
    < rimuovi il processo da S.L>
    wakeup(P)
    // c'è almeno un processo in coda risvegliamolo
        end;
end;
```

L'operazione sleep sospende il processo che lo invoca. L'operazione wakeup riavvia l'esecuzione del processo bloccato da *P.* Il valore di *S* potrà essere negativo e indica il numero di processi in attesa al semaforo. Ciò è la conseguenza del decremento introdotto nella funzione *P.* L'aggiornamento della lista avviene seguendo una filosofia *FIFO (First in first out)* tipica delle code, ossia il primo ad entrare e il primo ad uscire. Per attivare la regione critica è sufficiente usare:

```
P(semaforo)
<regione critica>
V(semaforo)
```

STATEMENT REGIONE CRITICA

Un'ulteriore evoluzione è stata apportata da Brinc Hansen e Hoare, nel lontano 1972 una variabile *x* di tipo *T,* condivisa da più processi deve essere dichiarata tale:

```
var x:shared T
```

Per accedere alla variabile bisogna usare lo statement *regione critica.*

```
region x to S
```

Il significato dovrebbe essere chiaro. Mentre, l'istruzione *S* è in esecuzione, altri processi non possono accedere alla variabile *x.* Facciamo subito un semplice esempio.

```
region x to S1;
region x to S2;
```

Le due istruzioni sono eseguite concorrentemente in distinti processi sequenziali. Il risultato che si otterrà è l'esecuzione sequenziale; prima una e poi l'altra. In altri termini o *S1* segue *S2,* oppure *S2* segue *S1.* Supponiamo di avere una variabile strutturata che sia usata dal sistema. Quale migliore esempio di uno stack, giacché il SO ne fa uso massiccio. Volendo essere pignoli potremmo parlare di una classe che oltre alla struttura renda disponibili funzioni di manipolazione come *push* e *pop.* Ma l'obiettivo attuale è soltanto osservare come si possa usare il nuovo costrutto.

```
Var stack : shared record
     vett:array[1..100] of integer;
     top: 0..100;
            end;
Procedure entry
Begin
  region stack do
  begin
  …..
  end
End.
```

I puntini indicano le operazioni di manipolazione che applichiamo alla variabile *stack.*

IMPLEMENTAZIONE DELLE REGIONI CRITICHE

Analizziamo come si possa realizzare una regione critica a partire da costrutti di base. Si suppone che siano disponibili nel SO le primitive *semafori, P* e *V.* A partire da esse costruiremo le regioni critiche.
Per ogni variabile *x* di tipo *"sharato"* (se preferite condivisa) il compilatore crea un semaforo inizializzato a *1.*

```
Var x_mutex : semaphore;
… x_mutex := 1;
```

Mentre lo statement regione critica

```
region x do S;
```

Viene sostituito dalla sequenza:

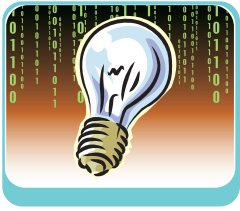
```
P(x_mutex)
   S;
V(x_mutex)
```

Le regioni critiche possono essere innestate.
Ecco come si usano:

```
Var x: shared Tx;
       y: shared Ty;
..
region x do region y do S
```

Mentre è eseguita l'istruzione *S* la variabile *y* è pro-

I TUOI APPUNTI

Utilizza questo spazio per le tue annotazioni

tetta e non può essere usata da altri processi. Mentre accade ciò che è descritto nel periodo precedente, (altro non è che un'istruzione), la variabile *x* non può essere usata da altri processi. Purtroppo, l'utile strumento delle regioni critiche innestate produce alcuni effetti indesiderati. Si può generare *deadlock*, ovvero lo spauracchio del programmatore concorrente. Capiamo come con un esempio.
Consideriamo le due regioni critiche innestate eseguite dai due processi *Q* e *W*.

```
Var x: shared Tx;
       y: shared Ty;
cobegin
         Q ->  region x do region y do SP
         W -> region y do region x do SQ
coend;
```

Se *Q* e *W* entrano quasi contemporaneamente nelle regioni critiche *x* e *y*, si avrà *deadlock*. Più avanti si comprenderà, nel caso specifico, il significato dell'accezione quasi. Osserviamo nei vari istanti di tempo l'implementazione mediante primitive *P* e *V* nel caso più sfortunato, che genera lo stallo.

```
t0 :  Q esegue P(x_mutex);
t1 :  W esegue P(y_mutex);
t2 :  W esegue P(x_mutex), W attende giacché
                                          x_mutex è 0;
t3 :  Q esegue  P(y_mutex), Q attende giacché
                                          y_mutex è 0;
```

I due processi rimangono in attesa illimitata visto che per sbloccarli è necessaria l'azione dell'altro processo che a sua volta è in attesa. *Stallo* quindi.
La soluzione al problema che si presenta solo nella tipologia esaminata, di coppie di regioni critiche innestate, si può attuare imponendo un ordine. Se, ad esempio la regione *y* è innestata nella regione *x* allora si stabilisce che *y* è minore di *x*. Qualora si riscontri una tale relazione d'ordine non si permetterà l'immediata esecuzione di regioni critiche innestate al contrario (*x* innestata in *y*). Così viene evitato il *deadlock*.

DEADLOCK
Anche conosciuto come stallo è lo stato in cui possono trovarsi due o più processi. In particolare per due processi si innesca quando un processo in attesa *(wait)* attende il verificarsi di un evento del secondo processo che a sua volta è in *wait* e aspetta il verificarsi di un evento del primo processo. L'attesa può essere circolare nel caso di più processi.

REGIONI CRITICHE CONDIZIONALI

È una evoluzione del costrutto precedente che permette l'implementazione della mutua esclusione, ma soprattutto della sincronizzazione. La sintassi è la seguente:

```
var x: shared T;
region x: when EC do S
```

Con *EC* espressione condizionale che può assumere uno tra due valori, vero o falso. Inizialmente si cattura la variabile x e poi si valuta, in muta esclusione, se *EC* è vera; in tal caso si esegue lo statement *S*. Se *EC* è falsa si rilascia la mutua esclusione e si rimane in attesa che l'espressione condizionale diventi vera per poi ricominciare. Ci si potrebbe chiedere chi fa diventare vera *EC*. Certamente non lo stesso processo, poiché questo è fermo, bloccato in attesa che appunto *EC* diventi vera. Dovrà essere un altro processo, che cambierà lo stato di *EC* e permetterà al processo di eseguire *S*. Sviluppiamo un esempio di applicazione su un caso conosciuto. La struttura dati che si vuole condividere è un buffer implementato come array circolare.

```
Var buffer: shared record
                        vett: array[0..n-1] of T
                        cont, in, out: integer;
              end;
```

Le regioni critiche condizionali risolveranno i problemi di sincronizzazione. Scriviamo le due procedure associate al produttore di dati per il buffer e al consumatore. Il problema in questione, già esaminato in passato, prevede due attori: un produttore che inserisce un dato *d* all'interno del buffer e un consumatore che preleva un dato dal buffer. Entrambi gli attori avranno degli indici di riferimento nel buffer, in per il produttore, out per il consumatore. Attraverso l'operatore di resto mod si rende il vettore circolare, ovvero l'elemento successivo all'ultimo è il primo. Ecco il codice che permette la programmazione concorrente con le regioni critiche condizionali.

```
(* Produttore *)
Repeat
  <produce dato>
  region buffer when buffer.cont<n
  do begin
     buffer[n]:=d;
     in:=(in+1) mod n;
     cont:=cont+1;
        end;
Until false;
(* Consumatore *)
Repeat
  region buffer when buffer.cont > 0
  do begin
     d:= buffer[out];
     out:=(out+1) mod n
     cont:=cont-1
        end;
    <consuma dato>
Until false;
```

Inizialmente il buffer è vuoto. Se parte il consumatore si bloccherà sulla condizione di falso, espressa

in funzione del buffer. In tal caso il buffer viene rilasciato e il processo riprende. Con il campo cont si tiene traccia del numero di elementi presenti nel buffer. Quando il produttore genererà un dato una volta rilasciata la sezione critica il consumatore in attesa potrà utilizzarlo, in quanto la sua espressione condizionale risulterà vera. L'aspetto negativo sta nel fatto che ogni qual volta lo statament esce dalla regione critica bisogna rivalutare le espressioni di tutti i processi in attesa (in questo caso solo quello del produttore). Bisogna trovare che l'espressione condizionale di un altro processo sia vera. Ciò ha un considerevole costo di tempo. Quindi i due processi sono sincronizzati.

IMPLEMENTAZIONE DELLE REGIONI CRITICHE

Come per le regioni critiche, anche per le condizionali capiamo come verranno implementate dal SO, sempre con riferimento alle primitive *P* e *V*. Si vuole tradurre l'istruzione di base delle regioni critiche condizionali.

```
region x when EC do S
```

Per ogni variabile shared il compilatore riserva alcune strutture dati. Il semaforo *x_mutex* garantisce la mutua esclusione sulla variabile *x*. Il semaforo *x_wait* serve a bloccare i processi la cui espressione condizionale *EC* sia falsa. La variabile *x_count* è un contatore di processi bloccati sul semaforo *x_wait*, inizialmente vale zero poiché nessun processo è in attesa. La variabile *x_temp* viene utilizzata per la rivalutazione delle espressioni.

```
var x_mutex: semaphore;
       x_wait :semaphore;
       x_count, x_temp: integer;
… (* inizializzazione *)
x_mutex:=1;
x_wait:=0;
x_count:=0;
x_temp:=0;
```

L'implementazione della regione critica condizionale sarà la seguente

```
… P(x_mutex)
if not EC then
begin
   x_count:=x_count+1;
   V(x_mutex);
   P(x_wait) ;
   while not EC do
   begin
      x_temp :=x_temp+1 ;
      if x_temp < x_count then V(x_wait)
      else V(x_mutex)
   end;
   x_count:=x_count-1
end;
S;
if x_count>0 then begin
   x_temp:=0;
   V(x_wait)
end;
else V(x_mutex);
…
```

Questa implementazione assume una logica FIFO per l'esame dei processi in coda su un semaforo. Con *P(x_mutex)* si garantisce la mutua esclusione. Se *EC* è vera si può eseguire lo statement *S*. Se invece è falsa bisogna aggiungere un processo in attesa su *x_wait;* quindi incrementare *x_count*. Inoltre si può liberare la mutua esclusione con una *V(x_mutex)* visto che per il momento non si può occupare *x*. Successivamente, con il ciclo di *while* si attende che la condizione *EC* diventi vera. La variabile *x_temp* tiene traccia del numero di processi rivalutati. Tale variabile deve essere incrementata per indicare che sta per essere valutata nuovamente la condizione. A questo punto se *x_temp* è uguale a *x_count* vorrà dire che nessun processo si può risvegliare per cui si libera la sezione critica con *V(x_mutex)*, altrimenti vi sono altre possibilità per cui si risveglia *x_wait* con l'apposita *V*. In definitiva ogni processo risveglia il successivo tranne l'ultimo che si blocca. Una volta che l'espressione condizionale risulta vera si può decrementare *x_count*. La parte dopo lo statement *S* serve a risvegliare la valutazione dell'espressione *EC* nei processi bloccati.

CONCLUSIONI

È interessante osservare come la programmazione concorrente si sia evoluta. Le regioni critiche e la loro estensione condizionale hanno fornito il primo e importante strumento di programmazione ad alto livello. Gli attuali costrutti, usati in linguaggi come Java fanno uso delle idee che stiamo esaminando. Inoltre, si appoggiano a primitive che trattano la programmazione concorrente a basso livello, come i semafori. Insomma, un ventaglio completo che si completerà con il suo ultimo elemento al prossimo numero, quando verranno esaminati i conosciuti monitor. Dopo potremo facilmente dedicarci alla programmazione vera e propria di ambienti concorrenti. Acquisire le nozioni sintattiche di un qualsiasi linguaggio ad alto livello che attua la concorrenza sarà un gioco da ragazzi! Per cui non perdete la prossima puntata. Vi aspetto!

Fabio Grimaldi

COBEGIN… COEND

È uno dei primi costrutti apparsi sulla scena per attuare la programmazione concorrente. Fu introdotto da *Dijkstra;* la sua sintassi è:

```
cobegin
   S1, S2, … Sn
coend
```

Ogni singolo *statement* Si incluso all'interno di *cobegin … coend,* verrà eseguito in modo concorrente.

Biblioteca

ON LINE

UNA RAGNATELA DI CODICE

Developer.com è uno dei siti più longevi della storia di Internet rispetto dedicato ai programmatori. Contiene centinaia di news, link, righe di codice dedicate a tutti i linguaggi. Una risorsa decisamente utile soprattutto per chi vuole rimanere aggiornato sulle tecniche, anche sperimentali, che rinnovano il mondo della programmazione.
http://www.developer.com

OPENSOURCECMS

I Content Management Systems sono diventati ormai uno strumento diffusissimo. Da PHPNuke a Mambo, ormai la scelta è diventata estremamente vasta. OpenSourceCms recensisce praticamente tutti i cms opensource oggi disponibili, mostrando pregi e difetti di ciascuno.
http://www.opensourcecms.com/

4 GUYS FROM ROLLA

Un sito ricco di contenuti per tutti i programmatori Asp e Asp.NET. Contiene risorse molto tecniche che affrontano problemi di programmazione, come anche di sistemistica in ambiente Microsoft.
http://aspnet.4guysfromrolla.com/

RIVOLUZIONARIO PER CASO

Se avete tempo e pazienza potrebbe divertirvi fare una piccola ricerca in rete alla caccia delle origini del sistema operativo Linux. Senza troppa difficoltà rintraccerete il primo messaggio pubblico di un timido Linus Torvalds che, come il più classico dei Nerds, annunciava su un NewsGroups di avere iniziato a lavorare su un proprio sistema operativo derivato da un dialetto di Unix. Poco o niente funzionava ma il ragazzo si divertiva e questo era quello che contava. A distanza di più di 15 anni Linux è diventato il sistema di riferimento per l'OpenSource, composto da centinaia di moduli, vede la compartecipazione di qualche migliaia di programmatori allo sviluppo del solo Kernel, per non parlare poi dei milioni di persone che ogni giorno lo usano e contribuiscono a farlo crescere. Linus Torvalds ci racconta in questo libro la sua storia, di ragazzo "Nerd" topo di laboratorio, che con la sua curiosità ha posato nelle fondamenta della storia dell'informatica uno dei mattoni che ne stanno determinando l'evoluzione.

Difficoltà: Bassa • **Autore:** Linus Torvalds e David Diamond • **Editore:** Garzanti • **ISBN:** 88-11-67859-5 • **Anno di pubblicazione:** 2001 • **Lingua:** Italiana • **Pagine:** 285 • **Prezzo:** € 9,50

HACKERS LA STORIA DELLE STORIE

Divertente, intrigante, scorrevole, curioso questo libro di "Maya" per la casa editrice "Malatempora". Maya: "Un corpo italiano, un'anima nera, un cuore messicano, una mente indù, una libido cybernetica, una residenza nel cyperspace. Razza Klingon" questo è quanto si legge di lei sull'ultima pagina del libro "Hackers: la storia delle storie". Descrizione affascinante per Maya Checchi, autrice di questo libro che si occupa della storia degli Hacker. Il volumetto si snoda in 112 pagine che scivolano quasi senza accorgersene fra sorrisi e stupore che si dipingono sul volto di chi legge man mano che la lettura va avanti. Maya ci racconta la storia dell'Hacking da quello italiano a quello internazionale e persino di quello femminile, sconosciuto ai più, ma non meno interessante. Non si può fare di ogni Hacker un eroe ma certamente l'Hacking inteso come la volontà di conoscere di capire, di oltrepassare ogni limite nel tentativo di vincere una sfida con se stessi e con gli altri rappresenta senza dubbio un'avventura non priva di fascino.

Difficoltà: Bassa • **Autore:** Maya • **Editore:** Malatempora • **ISBN:** 88-8425-001-3 • **Anno di pubblicazione:** 2002 • **Lingua:** Italiana • **Pagine:** 112 • **Prezzo:** € 8

LA SICUREZZA DELLE RETI WIRELESS

Il mondo senza fili è entrato prepotentemente nelle nostre case e nei nostri computer. Se da un lato questo rappresenta una comodità senza paragoni e ci svincola da lacci e lacciuoli derivanti dalla fisicità delle connessioni, dall'altro porta alla ribalta innegabili problemi di sicurezza legati alla propagazione a mezzo etere dei dati. In America è già diffuso il War-Driving. Malintenzionati vanno a spasso con portatile e scheda Wireless sotto braccio alla ricerca di reti aperte che gli consentano di fare breccia nelle difese aziendali o nei computer dei meno accorti. Il libro edito da Apogeo per le penne di Merrit Maxim e David Pollino affronta dettagliatamente tutte le tematiche

legate alla protezione delle reti Wireless, senza tralasciare gli aspetti pratici dell'implementazione ed occupandosi comunque di protocolli e teoria della comunicazione. Un volume completo, interessante e utile per tutti coloro che vogliono prepararsi alle sfide della comunicazione della nuova era senza incorrere nello sgambetto di qualche persona con pochi scrupoli nell'approfittare dei buchi lasciati liberi da una rete poco protetta.

Difficoltà: Alta • **Autore:** Merrit Maxim, David Pollino • **Editore:** Apogeo - lo trovate anche con "le grandi guide di ioProgrammo"